



NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

BENCHMARKING AND ANALYSIS OF THE SRC-6E
RECONFIGURABLE COMPUTING SYSTEM

by

Kendrick R. Macklin

December 2003

Thesis Advisor: Douglas Fouts
Co-Advisor: Theodore Lewis

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2003	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Benchmarking and Analysis of the SRC-6E Reconfigurable Computing System			5. FUNDING NUMBERS	
6. AUTHOR(S) Kendrick R. Macklin				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) This thesis evaluates the usefulness of the SRC-6E reconfigurable computing system for a radar signal processing application and documents the process of creating and importing VHDL code to configure the user definable logic on the SRC-6E. A false-target radar-imaging algorithm was chosen and implemented on the SRC-6E. Data from alternative computational approaches to the same problem are compared to determine the effectiveness of SRC-6E solution. The results show that the implementation of the algorithm does not provide an effective solution when executed on the SRC-6E. An evaluation of the SRC-6E difficulty of use is conducted, including a discussion of required skills, experience and development times. The algorithm test code and collected data are included as appendices.				
14. SUBJECT TERMS Benchmark, Reconfigurable Computing, VHDL, SRC-6E, FPGA, False Radar Target Synthesis			15. NUMBER OF PAGES 149	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**BENCHMARKING AND ANALYSIS OF THE SRC-6E RECONFIGURABLE
COMPUTING SYSTEM**

Kendrick R. Macklin
Lieutenant, United States Naval Reserve
B.S., San Diego State University, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
December 2003**

Author: Kendrick R. Macklin

Approved by: Douglas Fouts
Thesis Advisor

Ted Lewis
Co-Advisor

John P. Powers
Chairman
Department of Computer and Electrical
Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis evaluates the usefulness of the SRC-6E re-configurable computing system for a radar signal processing application and documents the process of creating and importing VHDL code to configure the user definable logic on the SRC-6E. A false-target radar-imaging algorithm was chosen and implemented on the SRC-6E. Data from alternative computational approaches to the same problem are compared to determine the effectiveness of SRC-6E solution. The results show that the implementation of the algorithm does not provide an effective solution when executed on the SRC-6E. An evaluation of the SRC-6E difficulty of use is conducted, including a discussion of required skills, experience and development times. The algorithm test code and collected data are included as appendices.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	PURPOSE	1
B.	FALSE TARGET RADAR IMAGING ALGORITHM	1
C.	FALSE-TARGET RADAR-IMAGING CHIP DESIGN	2
D.	FALSE-TARGET RADAR-IMAGING PROGRAM DESIGN	5
E.	REMAINING CHAPTER OUTLINE	7
II.	SRC-6E ARCHITECTURE AND SOFTWARE ENVIRONMENT	9
A.	INTRODUCTION	9
B.	SRC-6E HARDWARE OVERVIEW	9
C.	SOFTWARE ENVIRONMENT	11
	1. Operating System	11
	2. Programming Environment	11
D.	MAJOR DOCUMENTATION	12
	1. SRC-6E C Programming Environment Guide	12
	2. SRC-6E Fortran Programming Environment Guide	12
	3. SRC-6E MAP Hardware Guide	12
	4. SRC-6E MAP Macro Developers Guide	12
	5. Macro Data Sheet Library	13
III.	DEVELOPMENT AND TESTING IN VHDL WITH ALDEC ACTIVE-HDL	
	5.2	15
A.	INTRODUCTION	15
B.	FUNCTIONAL BLOCKS	15
	1. D-Type Flip Flops	15
	2. Adders	15
	3. Look-Up Table (LUT)	16
	4. Control Logic Block (CLB)	16
	5. Gain Shifter	16
	6. One Range Bin	18
	7. Two Range Bins	18
	8. Four Range Bins	18
IV.	PORTING THE VHDL CODE TO THE SRC-6E	19
A.	INTRODUCTION	19
B.	THE SRC-6E FILE TYPES	19
	1. .info	19
	2. .box	19
	3. .mc	20
	4. .c	20
	5. makefile	20
	6. .vhd	20
	7. Other Types	20

C.	CODE DEVELOPMENT	21
1.	Version 1.0	21
2.	Version 1.1	21
3.	Version 1.2	22
4.	Version 2.0	22
5.	Version 2.1	22
6.	Version 2.2	23
7.	Version 2.3	23
8.	Version 2.4	24
9.	Version 2.5	24
10.	Version 3.0	25
D.	SYNTHESIZABILITY	25
1.	Gain Shifter Changes	25
2.	LUT Changes	26
E.	TIMING FAILURES	26
1.	Single 8-bit CLAH	27
2.	Three 4-bit CLAH	28
3.	Two 4-bit and one 8-bit CLAH	28
F.	MEMORY ALLOCATION CHANGES	28
V.	DATA COLLECTION AND TIMING ANALYSIS	31
A.	INTRODUCTION	31
B.	BENCHMARK TEST PLATFORMS	31
1.	C Program Executed on a Windows-based Machine	31
2.	C Program Executed on the SRC-6E	31
3.	VHDL Code on the SRC-6E MAP	31
C.	TIMING DATA COLLECTION METHOD	32
D.	TIMING DATA ANALYSIS	33
1.	Methods	33
2.	Results	33
3.	Conclusions	37
VI.	CONCLUSIONS	39
A.	INTRODUCTION	39
B.	DIFFICULTY OF USE	39
1.	Necessary Skills	39
2.	Experience Level	39
3.	Development Time	40
C.	APPROPRIATENESS OF THIS ALGORITHM	40
D.	RECOMMENDATIONS FOR FUTURE WORK	40
1.	Develop Implementation of More Range Bins. ...	40
2.	Develop a More User-Friendly Programming Environment.	41
3.	Testing Other Applications.	41
APPENDIX A	43
A.	CHIP2_SIM.C	43

APPENDIX B	51
A. D-TYPE FLIP FLOP	51
B. 5-BIT REGISTER	51
C. 8-BIT REGISTER	52
D. 13-BIT REGISTER	53
E. 17-BIT REGISTER	54
F. FULL ADDER	55
G. FULL ADDER WITH OVERFLOW SIGNAL	55
H. 5-BIT ADDER	56
I. 16-BIT ADDER WITH OVERFLOW SIGNAL	56
J. LUT	57
K. CONTROL LOGIC BLOCK	58
L. SHIFTER	59
M. ONE RANGE BIN	60
N. TWO RANGE BINS	63
O. FOUR RANGE BINS	64
APPENDIX C	67
A. MACRO VHDL FILE	67
B. MAKEFILE	84
C. MACRO INFO FILE	85
D. MACRO BLACKBOX FILE	86
E. C DRIVER PROGRAM	86
F. MAP CODE FILE	92
G. SAMPLE PHASE SAMPLE INPUT FILE	93
H. SAMPLE RANGE BIN GAIN INPUT FILE	93
I. SAMPLE SCREEN OUTPUT	94
J. SAMPLE OUTPUT DATA FILE	94
K. SAMPLE RANGE BIN PHASE ROTATION INPUT FILE	95
APPENDIX D	97
A. SRC-6E MACRO DATA	97
B. SRC-6E C PROGRAM DATA	119
C. WINDOWS C PROGRAM DATA	121
LIST OF REFERENCES	125
INITIAL DISTRIBUTION LIST	127

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. False Target Radar Imaging Algorithm Usage	2
Figure 2. False Target Radar Image Chip Signal Flow	3
Figure 3. Internal Design of the Control Logic	5
Figure 4. Signal Flow for Four Cascaded Range Bins	6
Figure 5. SRC-6E System Diagram (After Ref. 2.)	9
Figure 6. MAP Interface Block Diagram (From Ref. 2.)	10
Figure 7. 16-bit Adder Versions	27
Figure 8. Comparison of Average Total Time	33
Figure 9. Semi-Log Comparison of Average Total Time	34
Figure 10. Comparison of Average Time per Sample	35
Figure 11. Semi-Log Comparison of Average Time per Sample ...	36
Figure 12. Comparison of Average Time per Sample	36

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	False-Target Radar-Imaging Program Example Using Four Range Bins	6
Table 2.	Gain Shifter Operational Data (After Ref. 9.)	17

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would like to thank Professor Russ Duren, formerly of the Naval Postgraduate School, but now at Baylor University, for donating his personal time and expertise to help me debug my code on several occasions. Without his assistance, the focus of this thesis would have been how I could not make the algorithm work on this computer as opposed to a presentation of benchmarks and results.

I would also like to acknowledge David Caliga of SRC Computers, Inc. for providing personal assistance in helping me to understand and use the SRC-6E computer. His help went well beyond what I would normally expect from that required by a technical support contract by personally debugging my code on several occasions.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The purpose of this research was to evaluate the performance, correctness, and ease of use of the SRC-6E reconfigurable computing system built by SRC Computers, Inc., and also to aid in establishing a broad base of knowledge on what types of applications are appropriate for implementation on this type of machine. To this end, it was necessary to first choose a readily available yet suitably complex algorithm for implementation on the SRC-6E. The algorithm chosen was based on a custom chip design previously developed by a faculty/student research team at the Naval Postgraduate School which creates false target radar images. A C language program, written by Professor Douglas Fouts, was also available to use as a standard for comparing the accuracy of results throughout the research.

Reconfigurable computing is defined as "the capability of reprogramming hardware to execute logic that is designed and optimized for a specific user's algorithms" [1]. The SRC-6E reconfigurable computer is a Linux-based system consisting of two independent sides labeled A and B which each contain motherboards holding dual Intel P3 Xeon 1-GHz processors, 1.5 gigabytes of memory, and a SNAP interface card. The SNAP card is a custom interface card which plugs into a motherboard DIMM memory slot and provides connections to the MAP board which is located in a third section of the system. A single MAP board consists of two independent MAPs. MAP, a registered trademark of SRC Computers, Inc., is the name for the custom hardware. Each MAP consists of three Xilinx Virtex-II-series XC2V6000 FPGAs and 24 megabytes of memory. One of the FPGAs is reserved for "control

logic" while the other two, available for user programs, are called "user logic". The memory is split into six equal banks, labeled A through F, of 4 megabytes each. The user FPGAs are connected to a fixed 100-MHz clock.

Code written in the hardware description languages Verilog and/or VHDL can be ported for use on the SRC-6E with only minor changes. Several support files are required to make the code target the user logic. These files primarily describe the interfaces to the code. The algorithm selected for the research described here was written in VHDL and converted for use on the SRC-6E.

In order to evaluate the effectiveness of the SRC-6E, timing data was collected from several sources. The first data source was the executable created on the SRC-6E which utilizes the reconfigurable user logic. The second data source was a C program which performs the same functionality as the VHDL code. This code was compiled and executed on a 3-GHz Pentium 4 system, utilizing 2 gigabytes of DIMM memory and the Windows XP Professional operating system. The third data source was the same C program running on the 1-GHz Xeon processor on the Linux based SRC-6E (but not using the MAP). Several input data sets were created for testing. Each individual input data value consists of a 5-bit number, written as two hexadecimal digits, which represent an intercepted radar signal. Data sets containing 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16284, 32786, 65536, 131072, 262144, and 500000 data values were used. Five timing runs were conducted for each data set on all three data sources.

The timing data shows the SRC-6E MAP execution time is extremely fast, even for very large data set sizes. However, the total execution time for the SRC-6E VHDL macro takes considerably longer than all other benchmark sources. The extra time represents delays in the system to prepare and transfer the data in and out of the MAP which cause the SRC-6E execution time to be longer for all input set sizes, initially by an order of magnitude.

As input set size is increased the timing results begin to converge. The overhead in the SRC VHDL macro clearly dominates the results for smaller sample set sizes. However, for larger sample set sizes, the overhead time is amortized over the total time to be nearly insignificant. Presumably, the SRC macro total execution time would eventually meet the other benchmark platforms if the sample set size could be further increased. However, this is not possible with the current macro design due to the memory design of the SRC-6E hardware.

Programming the SRC-6E to use user-defined macros requires knowledge of high-level programming languages, hardware description languages, hardware component design, and synthesizability. Relatively few people possess all of these skills to use the system effectively without first receiving significant training. However, programming the system using only high-level languages of C or Fortran is possible which widens the potential user base to many more people. More research needs to be performed to determine if either method produces more effective solutions.

The SRC-6E has a relatively steep learning curve. There are a few examples in the documentation and a very

small body of work in place using the system. The errors generated by the system during development are not intuitive and cannot be solved without previous experience with solving the same errors. There are no development tools in place to assist novice users in programming the system. More research is required to see how much experience on the system is required to prevent and or recognize these types of errors quickly.

The development time to implement solutions on this system appears to be high, primarily due to the steep learning curve and lack of development tools. More research must be performed to quantify the development time and see how it improves once a group of experienced repeat users is grown. No research has yet been performed with large projects, employing multiple programmers, to see if the total project time can be reduced effectively.

Since it is pipelined and supports parallel processing, the chosen implementation of the false-target radar-imaging algorithm appears to be one that would benefit from a reconfigurable computer. However, the current implementation has been shown to lack the necessary parallelism required to fully utilize the hardware and make it effective. Without increases in the memory size allocated for the user logic, the implementation on the SRC-6E is not an effective solution in terms of development time, processing time, or cost-effectiveness.

I. INTRODUCTION

A. PURPOSE

The purpose of this research was to evaluate the performance, correctness, and ease of use of the SRC-6E reconfigurable computing system built by SRC Computers, Inc., and also to aid in establishing a broad base of knowledge on what types of applications are appropriate for implementation on this type of machine. To this end, it was necessary to first choose a readily available yet suitably complex algorithm for implementation on the SRC-6E. The algorithm chosen was based on a custom chip design previously developed by a faculty/student research team at the Naval Postgraduate School which creates false target radar images. A C language program, written by Professor Douglas Fouts, was also available to use as a standard for comparing the accuracy of results throughout the research. This chapter discusses the basics of the false radar imaging algorithm, use of the chip design and C program in the research and gives an overview of the major steps required to implement and test the algorithm using the SRC-6E.

B. FALSE TARGET RADAR IMAGING ALGORITHM

The algorithm works by splitting a false target image into several range bins, as shown in Figure 1, where a ship is split into four range bins. Each range bin represents a portion of the vessel based on the distance from the radar source. Greater resolution can be achieved by having a greater number of range bins for a given false target. It can be observed from the geometry that the radar-signal travel distance is different for each range bin.

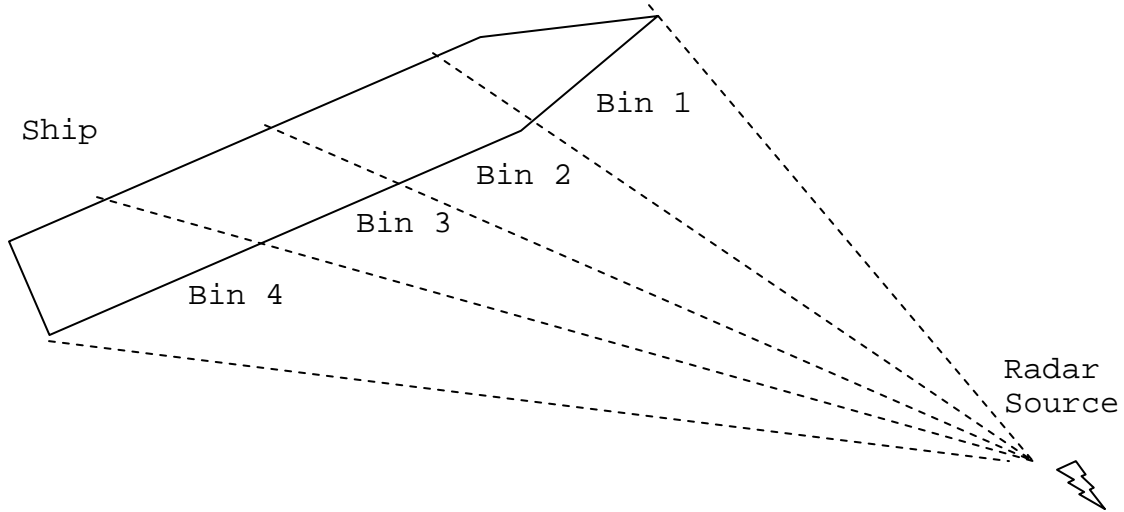


Figure 1. False Target Radar Imaging Algorithm Usage

Based on knowledge of a ship's radar image, an operator can set phase rotation and gain constants for each range bin. The algorithm begins with the interception and sampling of an interrogating radar pulse. The sample phase is then rotated by adding a rotation constant to it. Next, the sine and cosine are calculated. The gain is then applied to the results by multiplying by a gain value. The results of each range bin are then summed up to produce a radar reflection signal at a given time. With proper use, the ship can be made to appear in a false position, be of a different type of target, or to appear to be traveling with other ships.

C. FALSE-TARGET RADAR-IMAGING CHIP DESIGN

The false-target radar-imaging chip consists of a 6-stage pipeline which performs all necessary functions to create a false radar reflection for a single range bin. Figure 2 shows the signal flow through the slightly simplified version as was implemented during the research.



The basic steps of the algorithm are performed as follows:

1. The phase sample enters into register 3.
2. The phase rotation value enters into register 1, is then loaded into register 2, and is then added to the phase sample at adder 1. The results are then loaded into register 4.
3. The contents of register 4 enter the lookup table (LUT) and Sine and Cosine results are calculated. The remainder of the pipeline is split into two identical portions for each data result. The following steps outline the path for the Sine result.
4. The gain value enters at register 5, is then loaded to register 6, and proceeds to shifter 1 where it controls how the contents of register 7 are shifted before they proceed to register 9. This accomplishes modulo-2 multiplication.
5. The result from a preceding range bin enters at register 11 and is added to the contents of register 9 in adder 2 before proceeding to register 13.
6. The contents of register 13 are now available as output Q if this is the last range bin in the series or are sent to register 11 of a following range bin.

The control logic block receives signals URB (use range bin), PSVin (phase sample valid input), and ODVin (output data valid input). These signals are used to create the CLR13 (clear 13-bit register), CLR17 (clear 17-bit register), PSVout (phase sample valid output), and ODVout (output data valid output).

The internal design of the control logic is shown in Figure 3. The CLR13 and CLR17 signals are used to clear the register contents at the appropriate time in the pipelines when they do not contain valid data. This occurs during pipeline startup and shutdown. The PSVout signal is present to show the DRFM signal is valid. The ODVout sig-

nal is present to show that outputs Q and I contain valid data. The URB signal is present to allow the operator to disable a range bin. Figure 4 shows the signal flow when four range bins are connected together.

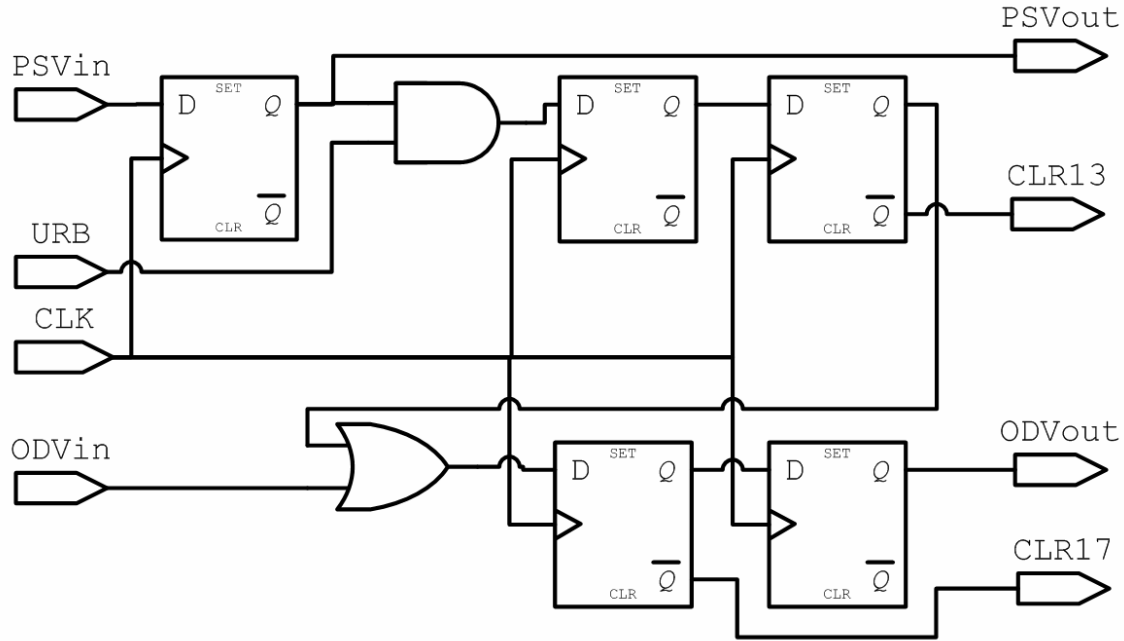


Figure 3. Internal Design of the Control Logic

D. FALSE-TARGET RADAR-IMAGING PROGRAM DESIGN

The false-target radar-imaging program was written in the C language. It performs the same arithmetic calculations as the false radar imaging chip but uses nested loop iterative structures instead of pipelines. While the chip requires a separate pipeline for each range bin, the program simply adds additional length to the appropriate arrays, trading off memory utilization for computational logic. Table 1 shows how the results of each of four range bins with an input of N samples are placed into the two dimensional array created by the program. Each row of the table is then summed up to produce the false target radar signal results.

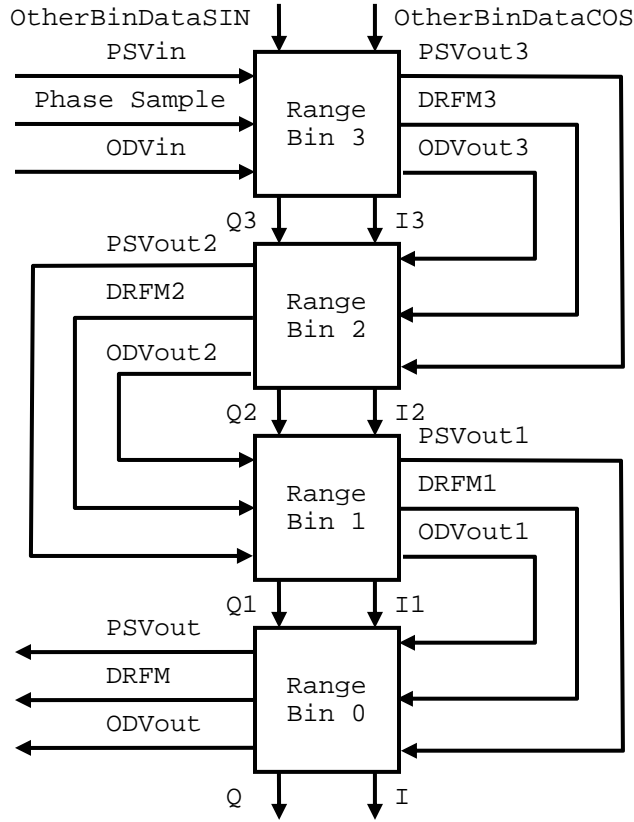


Figure 4. Signal Flow for Four Cascaded Range Bins

Bin 0	Bin 1	Bin 2	Bin 3
Sample 1 Results	0	0	0
Sample 2 Results	Sample 1 Results	0	0
Sample 3 Results	Sample 2 Results	Sample 1 Results	0
Sample 4 Results	Sample 3 Results	Sample 2 Results	Sample 1 Results
...
Sample N Results	Sample N-1 Results	Sample N-2 Results	Sample N-3 Results
0	Sample N Results	Sample N-1 Results	Sample N-2 Results
0	0	Sample N Results	Sample N-1 Results
0	0	0	Sample N Results

Table 1. False-Target Radar-Imaging Program Example Using Four Range Bins

The program was used as both a trusted source for results to test the research against as well as used in the timing comparisons discussed in Chapter V. The full code for the program can be viewed in Appendix A.

E. REMAINING CHAPTER OUTLINE

The following outlines the remaining chapters which roughly follow the major steps that were taken throughout the research:

- Chapter II discusses the SRC-6E architecture, programming environment, and documentation.
- Chapter III discusses programming the chip design using VHDL.
- Chapter IV discusses porting the VHDL code to SRC-6E environment
- Chapter V presents the data collection methods and analysis.
- Chapter VI provides conclusions and future work recommendations.
- Appendix A contains the modified C program originally written by Professor Douglas Fouts which was used as a standard for output correctness and as a source of timing data.
- Appendix B contains the final version of the VHDL code that was tested before porting to the SRC-6E.
- Appendix C contains the final version of the files used on the SRC-6E, including sample input and output.
- Appendix D contains all of the timing data collected during the research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. SRC-6E ARCHITECTURE AND SOFTWARE ENVIRONMENT

A. INTRODUCTION

This chapter provides a brief overview of the hardware, software, and documentation, of the SRC-6E reconfigurable computing system. Reconfigurable computing is defined as "the capability of reprogramming hardware to execute logic that is designed and optimized for a specific user's algorithms" [1].

B. SRC-6E HARDWARE OVERVIEW

The SRC-6E computer consists of two independent Linux computers (labeled A and B) and a MAP board, (see Figure 5).



Figure 5. SRC-6E System Diagram (After Ref. 2.)

MAP, a registered trademark of SRC Computers, Inc., is the name of the custom reconfigurable hardware. Each independent Linux computer contains a motherboard holding dual In-

tel P3 Xeon 1-GHz processors, 1.5 gigabytes of memory, and a SNAP interface card. The SNAP card is a custom interface card which plugs into a motherboard DIMM memory slot and provides connections to the MAP board which is located in the MAP Chassis. A single MAP board consists of two independent MAPs. A block diagram of a single MAP is shown in Figure 6. A MAP consists of three Xilinx Virtex-II-series XC2V6000 FPGAs and 24 megabytes of memory (labeled OBM on Figure 6).

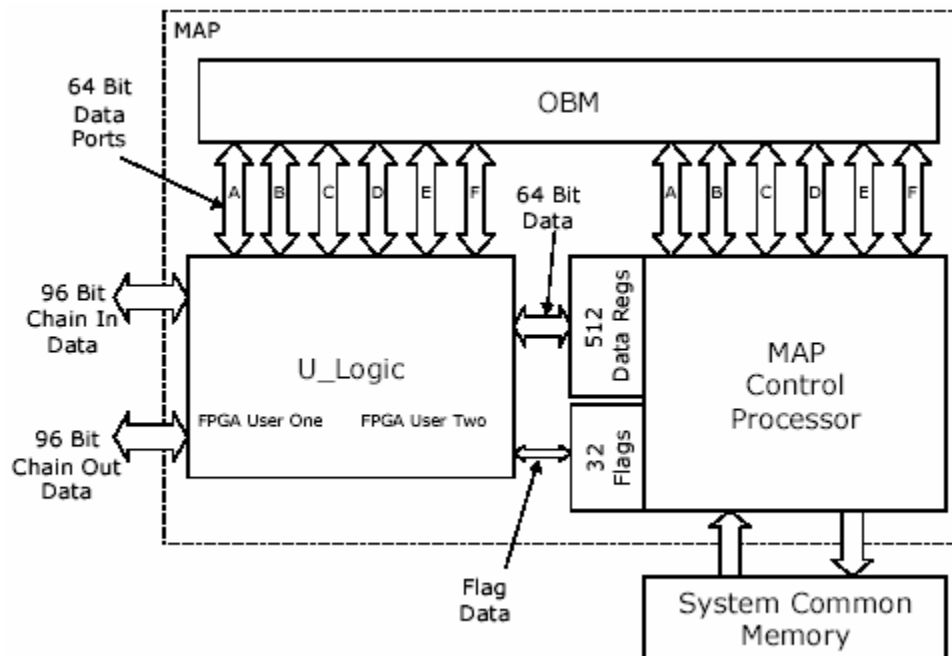


Figure 6. MAP Interface Block Diagram (From Ref. 2.)

One of the FPGAs is reserved for "control logic" while the other two, available for user programs, are called "user logic". The OBM memory is split into six equal banks, labeled A through F, of 4 megabytes each. The user FPGAs are connected to a fixed 100-MHz clock, which seems overly restrictive. According to Xilinx product specification sheets, the Virtex-II-series FPGAs can run at clock speeds as low as 1 MHz and upwards of 400 MHz [3]. Programmer

control of the clock speed on the SRC-6E would make the system more flexible. Each MAP also has a chain port which can be used for direct I/O to the user logic, but was not used during this research.

C. SOFTWARE ENVIRONMENT

1. Operating System

The operating system for the SRC-6E is Red Hat Linux, which has been augmented with custom drivers and libraries to support the MAP and SNAP hardware. The built-in graphical text editor in Linux is called GEdit. Programmers experienced with UNIX can use the standard line type text editors such as VI if they choose. Both contain the minimal functionality required of a text editor to write the required files for the SRC-6E.

2. Programming Environment

The programming environment for the SRC-6E is called Carte. Carte allows a user to write code in a high level language, either C or Fortran, that directly targets the user programmable FPGAs in the MAP. In addition, users can write their own "macros" using the hardware definition languages Verilog and/or VHDL. At compile time, all user code and macros are linked together into a single executable file. Carte includes standard compilers for the Intel microprocessors as well as custom MAP compilers for both Fortran and C. Synplify Pro software by Synplicity, Inc. is used for FPGA place and routing. This program normally runs under Windows version but is executed in the Linux environment using a Windows emulator called Wine.

Since Carte relies on the built-in Linux editors, the SRC-6E programming environment does not have any of the modern features a programmer expects from editors available

in products such as Microsoft's Visual C++ or Borland's J-Builder. Lack of syntax and error checking in the programming environment is a serious drawback when using this system. Some error messages are produced at compile time, but they are cryptic at best, especially for someone not used to the Linux environment. There are several file types which must interact during the compile process, as will be discussed in Chapter IV. The intricate details of these files can be quite confusing and it is often difficult to identify which file contains the problem based on the error messages given at compile time. Rudimentary checking of these files within a custom editor would greatly improve the entire programming process.

D. MAJOR DOCUMENTATION

The documents discussed here come with the SRC-6E to aid in its programming.

1. SRC-6E C Programming Environment Guide

Driver code must be developed to create the interface to the user logic. This document describes how to write this code using the C language [4].

2. SRC-6E Fortran Programming Environment Guide

Similar to the C Programming Environment Guide, this document describes how to write similar code using the Fortran language [5].

3. SRC-6E MAP Hardware Guide

This document contains hardware implementation specifics of the MAP which are well below the level required for users to successfully program the SRC-6E [2].

4. SRC-6E MAP Macro Developers Guide

This document discusses general information on the use of the Macro Data Sheet Library, including naming conventions, interfaces, fanout and combinatorial delays [6].

5. Macro Data Sheet Library

The library contains data sheets for all macros developed by SRC for the SRC-6E. A list of all currently supported macros is available in a technical note, Ref. 7. The macros can be used like regular function calls in the chosen programming environment language (C or Fortran). The macros include all basic math and logic functions currently supported by the environment. There are also several support macros which include, among others, various macros for combining and splitting data structures.

This chapter provided an overview of the hardware, software and documentation of the SRC-6E computer. The next chapter will discuss development and testing of the VHDL code used in the research.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DEVELOPMENT AND TESTING IN VHDL WITH ALDEC ACTIVE-HDL 5.2

A. INTRODUCTION

This chapter describes the development of the false-target radar-imaging macro in VHDL before it was ported to the SRC-6E environment. This portion of the research was performed before receipt of the SRC-6E system or any training on the system was received. As a result, the macro that was originally developed contained the correct functionality but was not optimized for the SRC-6E environment. Development of the macro was performed in a Windows XP environment using Aldec Active-HDL 5.2 software.

B. FUNCTIONAL BLOCKS

The False-Target Radar-Imaging chip was implemented directly into VHDL by direct programming of the code. Each component of the design was created using separate functional blocks of VHDL code. Several of the basic building blocks of code were taken from Ref. 8 and modified as necessary. The code for this section can be viewed in Appendix A.

1. D-Type Flip Flops

The six pipeline stages required registers, which were implemented as D-type Flip Flops. Single-bit registers were designed that are loaded on the rising clock edge and have both enable and clear input signals. The 5-, 8-, 13-, and 17-bit registers required for the design were created by instantiating multiple copies of the single-bit registers.

2. Adders

A single-bit full adder was coded using the design of Ref. 8. The 5- and 16-bit adders required for the design were created by instantiating multiple copies of the single-

bit adder. A simple ripple carry design was used at this point in the research. Chapter IV will discuss why this was later modified with carry look-ahead circuitry. For the 16-bit adder, a special final single-bit stage was developed to propagate an overflow signal if generated by previous range bin stages.

3. Look-Up Table (LUT)

The LUT was originally developed starting with a design from Ref. 7, but was later heavily modified. The LUT takes a single 5-bit input and performs simultaneous look-ups using data from both sine and cosine tables. The output of the LUT is two 8-bit values, one each for sine and cosine. The initial design had the correct functionality but was later modified after porting to the SRC-6E. The required modifications will be discussed in Chapter IV.

4. Control Logic Block (CLB)

The CLB was created by instantiating several of the flip flops with some basic logic functions to create the design shown in Figure 3.

5. Gain Shifter

The shifter takes a 4-bit control input and shifts the 8 bits of input data into a 13-bit output. The shifter is designed to provide a maximum gain multiplication of 1024. However, applying this to an 8-bit input results in an 18-bit output with more dynamic range than is necessary [9]. Therefore, the least significant 5 bits are truncated to create a 13-bit output. Table 2 shows how the control bits affect the shift and the resulting resolution of the output.

Control Code	Multiplication Factor	Size of Shift	Sin/Cosine Wave Resolution
0	1	0	3
1	2	1	4
2	4	2	5
3	8	3	6
4	8	3	6
5	16	4	7
6	32	5	8
7	64	6	8
8	16	4	7
9	32	5	8
10	64	6	8
11	128	7	8
12	128	7	8
13	256	8	8
14	512	9	8
15	1024	10	8

Table 2. Gain Shifter Operational Data (After Ref. 9.)

Because the input data could be negative, it was also necessary to preserve the sign bit by copying it as necessary to the upper bits in the output. The original version of this code used a case statement and some simple math to determine which bits were shifted where. The version ran correctly in the Aldec simulation software, but required

modification when porting to the hardware, which will be discussed further in Chapter IV.

6. One Range Bin

A single range bin was created by instantiating the above parts and creating an appropriate interface. The code was tested by comparing the output to the C program run on the same data set. After some minor error correction to the lookup table entries, the code was incorrectly deemed to be correct. Additional testing later conducted with two range bins yielded additional errors in the CLB that were not found in the single range bin tests.

7. Two Range Bins

A system with two range bins was then created by instantiating two of the single range bins previously tested. Tests run on the same data sets with the C program yielded errors. As previously mentioned, problems were eventually discovered with the timing within the CLB. These problems were not identified while testing the single-range-bin since the CLB primarily creates signals to handle the interaction between multiple range bins. After correction of the errors, the output was deemed to be correct.

8. Four Range Bins

Finally, a system with four range bins was created by instantiating four of the single range bins with an appropriate interface. The signal flow of four range bins is shown in Figure 4. The code worked properly the first time. It was this version of the code that was initially ported to the SRC-6E.

This chapter discussed VHDL code development. The next chapter will discuss porting the code to the SRC-6E.

IV. PORTING THE VHDL CODE TO THE SRC-6E

A. INTRODUCTION

This chapter discusses the porting of the VHDL code to the SRC-6E and the required support files. Also discussed are changes that were required to the original code to make it compatible with the SRC-6E.

B. THE SRC-6E FILE TYPES

The process of writing code to target the user logic requires several file types. To import a user macro from either VHDL or Verilog, five files must be created: .info, .box, .mc, .c, and the makefile. Using only the last three, one can write code that targets the user logic without using a user defined macro. Examples of these file types can be viewed in Appendix C, which contains the final versions of all the files used.

1. .info

This file type is required whenever a user macro is used. It contains the following information:

- Macro name
- Macro type - stateful, external, and pipelined
- Latency - a number stating how many clock cycles before valid output is generated by the macro.
- List of inputs and outputs

The file type ".info" is a naming convention and is not required. Any filename can be used as long as it matches that listed in the makefile.

2. .box

This is another file type that is required only when using a user-defined macro. It is a Verilog style description of the input and output variables of the macro. The Verilog description is necessary for both VHDL and Verilog

macros. As with the .info type, the .box name is only by convention.

3. .mc

This file type is C code written to target the user logic. All code in this file will be implemented in hardware along with the user macro. Using this file type, it is possible to write code for the hardware using only the high-level language C without using any user-defined macros defined with a hardware description language.

4. .c

This file type is regular C code which provides the interface between the operating system and the hardware code defined in the .mc file. Code implemented in this file is executed on the Xeon processors.

5. makefile

This file is used by the command "make" when all the files are compiled and linked. It contains all of the file names and paths used, as well as the desired final executable name. Compiler flags and options can also be stated in this file.

6. .vhd

This file type is for VHDL macro files. In general, it is safest to merge multiple files into one. However, it is possible to build with separate files as long as they are listed in the proper order in the makefile. The compiler appears to be single pass so the files must be in the order they are used, with the lowest order file listed first.

7. Other Types

Two other file types can be used by users programming the user logic: .f, which is a Fortran file, and .v, which

is a Verilog file. These file types were not used during this research.

C. CODE DEVELOPMENT

Porting of the macro code began with creation of the required support files previously mentioned. Although the files are relatively small, creating them was non-trivial as there were no previous examples using VHDL macros. The process was a painful series of trial and error, particularly with the required contents of the .info and .box files. The code went through ten major revisions, with three major versions, over a period of about six months.

1. Version 1.0

The single-range-bin VHDL code was imported to the SRC-6E and all code modules were merged into a single .vhd file. The required support files were first generated using some unrelated examples in the C Programming Guide and a lot of guessing. The SRC data packing macros called combine and split were used to pack and unpack the data in the .mc file into two memory banks for input and one for output. Much trial and error was attempted on this version, but it would never make to create an executable.

2. Version 1.1

After discussion with SRC technical support, some new changes were tested. The .info and .box file format questions were mostly resolved in this version. The order of declarations within the .vhd file was changed to make the main macro appear as the top level to the compiler. The gain shifter code was modified to make it synthesizable. This version compiled to executable but caused unexplainable segmentation faults when run.

3. Version 1.2

In order to isolate the faults in this version, empty macros were made in VHDL consisting of only the interface information. After determining the problem was in the support files, the original VHDL macro was restored. Problems were isolated with misuse of the SRC packing macros and various other syntax errors. After much further work and testing, this version created a working executable which produced the proper output expected for a single range bin on a 32 sample size input.

4. Version 2.0

Encouraged by the success, a new version was created which attempted to implement four range bins. The SRC packing macros were not used in this implementation because they could not combine vectors shorter than 8 bits without wasting the remaining space. The VHDL macro uses 1-, 4-, 5-, and 17-bit signals. These odd sizes could not be efficiently combined with the pre-built macros and all packing of data was implemented in the C program, combining all input into two 64-bit words using a series of shifting and logic with masks. The VHDL macro interface was also modified to support the changes. This version created a working executable; however, some of the output data was incorrect.

5. Version 2.1

In order to help identify where the problems were, the output format was modified in the .c program to display the outputs of all four range bins. After several changes, the code began hanging when executed during the call to the MAP function. On recommendation of the SRC technicians, the method in which the array sizes were calculated was modified to ensure the arrays were properly padded and aligned

on 32-bit memory boundaries as required. The changes resolved the hanging problem but the output data was still incorrect.

6. Version 2.2

At this point, the researchers were stumped and searching for any possible reasons why the output data was wrong. The majority of the output was correct. The code generated several correct values followed by a single incorrect value. The remaining output was correct up until a certain point before the end of the data where it all went bad. Exploring all possibilities, it was discovered that the macro was failing the timing requirements to run within the 100-MHz clock. No errors or warnings were produced by the SRC environment to state this. The timing results are created along with many other files during the make process. For example, running the make process on Version 3.0 of this research generates 54 files split over 3 directories. Locating useful debugging information within these many files can be a chore. How the timing failures were resolved will be discussed later, but they ended up not being the problem.

7. Version 2.3

In order to troubleshoot the corrupt data problem, the 16-bit adder code was removed, which allowed the direct output of each of the four range bins to appear in the output. The data generated by each of the range bins showed the same general format of being mostly correct but all going bad after a certain point. Much attention was turned to the control logic at this point to see if it was the culprit but no errors could be found. To help isolate the problem, the current version of the VHDL code was exported back to the Windows environment and it produced the correct

output. At this point, the SRC-6E was incorrectly suspected to have either a software or hardware bug, possibly in the memory transfers. The software environment has a useful debugging mode called MAPTRACE which can be used to view the data before and after it is sent to the MAP. Observations of the file generated by MAPTRACE showed that the data was being passed to and received by the MAP correctly.

8. Version 2.4

This version still had the 16-bit adder removed. Minor changes to the LUT and gain shifter were implemented in this version to ensure that they were fully synthesizable but they did not affect the output. Troubleshooting with this version did not solve the problem but helped narrow the focus to the interface. Upon close examination of the interfaces it was noted that there were differences between the Windows version and the SRC version as to the way the data was packed in the SRC version. After exporting the packed data to the Windows version, the code produced the same identical faulty output as the SRC version. Since the two versions both produced the same identical output, it was determined that the problem had to be with the interface and input data.

9. Version 2.5

After closer inspection of the interface and the method used to pass in data, it was observed that the gain and phase shift signals were not being applied properly. This was an operational problem as the macro code was correct. Modifying how the signals were applied fixed the problem with the faulty outputs. At this point, the code was producing correct output and data collection was started on various sized data sets. While collecting the

data, it was noted that a segmentation fault would occur above certain array sizes.

10. Version 3.0

After discussion with SRC technical support, the code was modified to use a dynamic array allocation method which will be discussed later. The memory usage changes corrected the problem. All extra unnecessary output was also removed in this version. This final version was used to collect the data and is shown in Appendix C.

D. SYNTHESIZABILITY

Synthesizability is a style of hardware description language programming which allows the available layout tools to properly convert the code for hardware implementation on an FPGA. During the design of the code, the Aldec software was only used to simulate the VHDL code. Therefore, it only tested the code for functionality and did not consider if the code could actually be implemented in hardware. Two of the original code blocks, the gain shifter and LUT, required modification once ported to the SRC-6E so the layout tool could define them in hardware. The root cause of this was inexperience with both the VHDL language and the concept of synthesizability.

1. Gain Shifter Changes

The gain shifter went through two changes. Initially, the code was defined such that some of the variable bit widths were defined at run-time. This worked fine during emulation but could not be implemented in hardware. To make it work the code was written with a "case" statement that outlined specifically every possibility at runtime. Implementing this in hardware requires redundant logic and decoders to choose which portions to use during run time. Later, the code was streamlined again to remove an unneces-

sary function call which provided some savings in the final hardware definition. The function call, which converted data types from `bit_vector` to `integer`, also had an unnecessary variable length defined at run-time. When removing the variability, it was determined that the entire function was not required and the "case" statement was modified to incorporate the function's results directly.

2. LUT Changes

The same function call that was made in the Gain Shifter was also used in the LUT. Although this code worked properly, even with the variable length at runtime, the function call was unnecessary and similar methods were used to remove it from the code entirely. The removal resulted in a small space savings on the FPGA.

E. TIMING FAILURES

While debugging the code to determine the cause of some faulty output on the SRC-6E, it was noted that the macro was failing timing requirements for implementation with the 100-MHz clock. The worst path through the logic was reported to be in the portion of the pipeline that contained the 16-bit adder and that it exceeded the required time by 4.310 ns. The cause of the poor timing was that the 16-bit adder was initially implemented with a simple design using ripple carry propagation, shown in Figure 7a.

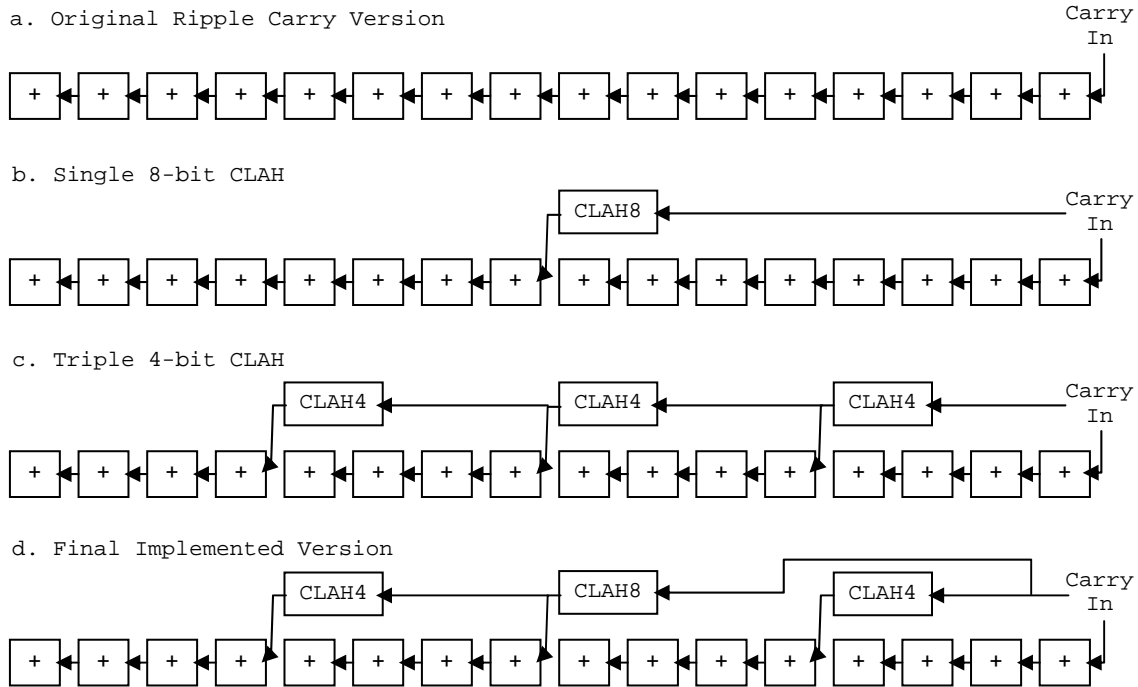


Figure 7. 16-bit Adder Versions

Several alternative designs were tested using carry look-ahead (CLAH) circuits to bring the delay time within that required for the 100-MHz clock. Of note, these modifications did not affect the output in any way and were not the solution to the problem being investigated at the time. The problem being investigated involved passing in improper input. Despite the fact that the timing was failing, the circuits were still working properly, demonstrating that there was possibly some error within the timing calculations or more likely that there was additional padding engineered within the design.

1. Single 8-bit CLAH

A single 8-bit CLAH circuit was designed and placed in the center of the carry chain, which effectively splits the chain in half as shown in Figure 7b. This improved the

time by almost 3 ns, but the circuit still failed timing by 1.615 ns.

2. Three 4-bit CLAH

A 4-bit CLAH circuit was designed and placed at three points in the carry chain. The circuit chained groups of four carries to each other, as shown in Figure 7c. This design slightly improved the timing but was still inadequate.

3. Two 4-bit and one 8-bit CLAH

Finally, combinations of 4-bit and 8-bit CLAH circuits were used, which effectively split the 16 carries into four pieces, as shown in Figure 7d. Initially, this design only improved the timing slightly which remained about 1.2 ns over what was required. Coincidentally, at the time of this testing, an upgrade to the Carte software was released, version 1.5. Remaking the same design after the upgrade created a result that was 0.401 ns under time. The reason why the new version of the software caused the timing improvement remains a mystery. No further modifications were made after this point.

F. MEMORY ALLOCATION CHANGES

Data passed into the MAP must be properly declared and aligned. There are two methods to accomplish this. The first method attempted used the SRC function "addr32." This method uses fixed sized arrays declared at compile time. The addr32 method worked fine up to fixed size arrays of 166,581 but caused segmentation faults when exceeding this value. A trial and error approach was used to determine the exact value at which the segmentation faults began. The number 166,582 has no apparent meaning when related to array sizes and is a very unusual number to fail on. Communication with SRC Computers, Inc. could not re-

solve why this occurs. However, using the second available method with the "cache_aligned_allocate" function allowed the array sizes to be declared correctly. This method uses run-time allocation to declare the proper array sizes and was tested successfully up to array sizes of 500,000 64-bit elements. Based on 4 megabytes of memory per bank, the theoretical limit is 524,288 64-bit values, but this upper limit was not tested.

This chapter discussed the necessary changes required to port the VHDL code to the SRC-6E environment. The next chapter will discuss benchmarking the SRC-6E, including data collection and analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DATA COLLECTION AND TIMING ANALYSIS

A. INTRODUCTION

This chapter discusses the benchmarks and methods used for collection of data and its analysis during the research.

B. BENCHMARK TEST PLATFORMS

1. C Program Executed on a Windows-based Machine

The C program shown in Appendix A was compiled and executed on a 3-GHz Pentium 4 processor system with 2 gigabytes of RAM running the Windows XP Professional operating system. The primary reason for this benchmark was to draw a comparison for cost-effectiveness between the high-cost special purpose SRC-6E system and a modern, off-the-shelf, general purpose computer.

2. C Program Executed on the SRC-6E

The same C program was compiled and run directly on the SRC-6E without using any of the custom hardware. Therefore, the data collected is based on the Linux operating system running on a 1-GHz Xeon 3 processor with 1.5 gigabytes of RAM. Although the system contains dual processors, only one thread is created while running the code and therefore it is believed that only one processor is utilized during the test. The primary reason for this benchmark was to test if the algorithm itself is suitable for implementation on the user-logic.

3. VHDL Code on the SRC-6E MAP

The VHDL user macro and support files (shown in Appendix C) were built and executed on the SRC-6E MAP. Two timing data results were collected from each of the runs, the total run time for the entire execution and the time of

execution on the MAP only. The two timing data results compare overhead time to actual execution on the MAP.

C. TIMING DATA COLLECTION METHOD

The input data sets were composed to represent a stream of intercepted radar samples. Each data item consists of two hexadecimal characters representing a five-bit intercepted radar sample. The 32-sample-size data set is shown in Appendix C, which represents the decimal numbers 0 to 31 in order. All other-sized sample sets were created by duplicating and repeating the same 32 samples in order. By doubling each previous sample set size the following set sizes were created: 32, 64, 128, 256, 512, 1,024, 2,048, 4,096, 8,192, 16,384, 32,768, 65,536, 131,072, and 262,144. The final set size of 500,000 was chosen as a convenient, large value that was close to the upper array size restriction allowed by the four megabytes of memory per bank on the SRC-6E.

Data from all test platforms were collected in order of increasing input set size. All raw data used in the timing analysis can be observed in Appendix D. The timing data was collected by running five consecutive runs of each input data set on each of the three benchmark platforms. The data for the Windows XP system were collected after a fresh reboot with all unnecessary programs closed. It should be noted that observation of the system usage during execution of the code showed that the processor and memory were not fully utilized. The reasons why the processor did not appear to be fully used and the methods Windows uses to measure performance are unknown. The SRC-6E system data were collected by running the executables on side A when no other users were using the system.

D. TIMING DATA ANALYSIS

1. Methods

The timing data are displayed in two types of graphs. The first is the average total time each test platform takes for each data set. The average is taken of the five data points for each input set size. The second is the average time per sample for each input set. First, the average is taken over the five data points and then it is divided by the input set size. All graphs are connected with straight line approximations between data points.

2. Results

Figure 8 shows the average total time vs. input set size for each of the four timing result sets.

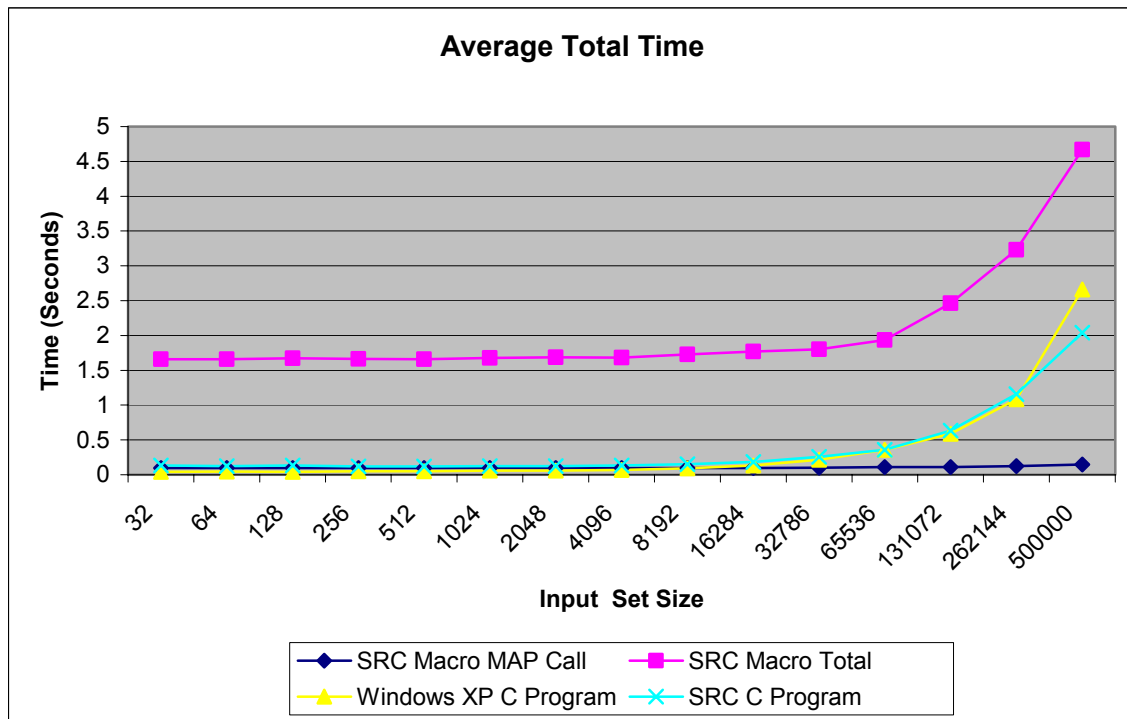


Figure 8. Comparison of Average Total Time

Figure 9 shows the same data displayed on a semi-log scale for better clarity in the lower sample set size region. All four curves are fairly constant up to the 16,284 sample

size. This result shows that, for small data set sizes, the overhead times inherent in the systems are much greater than the calculation times. We consider overhead to be all the data file read/write operations and memory accesses required to prepare the data for calculations. The SRC Macro MAP Call curve clearly shows the calculation time is insignificant compared to the total processing time.

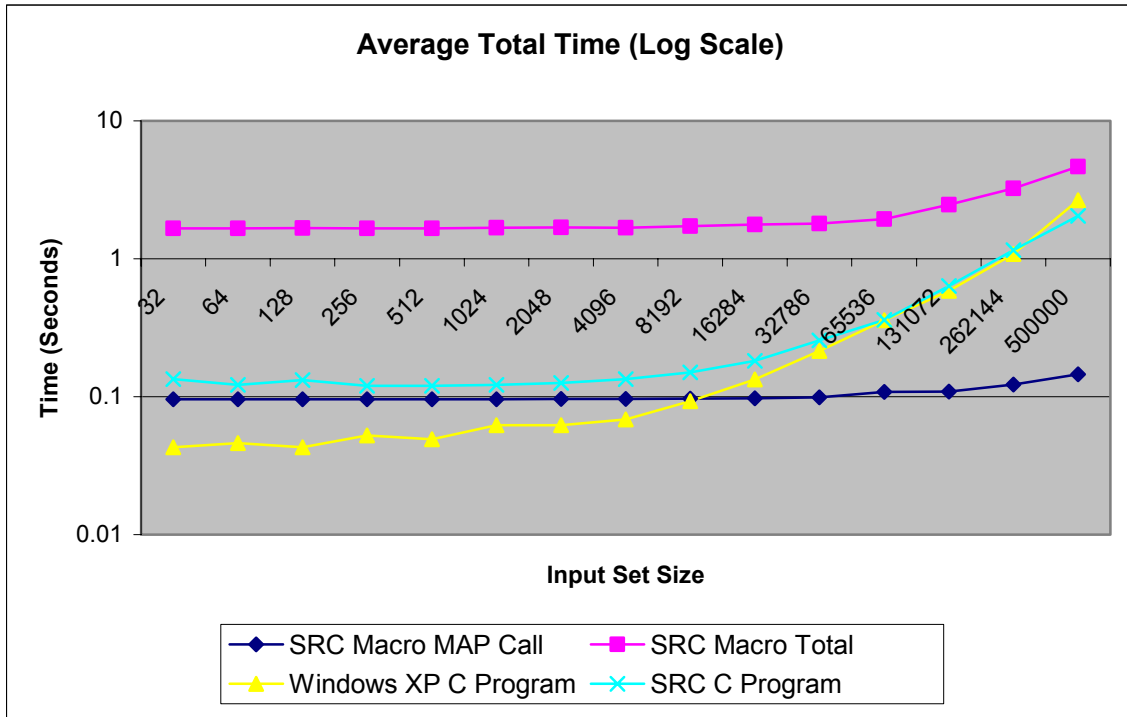


Figure 9. Semi-Log Comparison of Average Total Time

The SRC Macro MAP Call curve also shows the MAP execution time is extremely fast, even for very large data set sizes. However, the SRC Macro Total curve shows the total execution time for the VHDL macro takes considerably longer. The extra time represents delays in the system to prepare and transfer the data in and out of the MAP which cause the SRC execution time to be longer for all input set sizes, initially by an order of magnitude.

As the input set size is increased, we see the curves begin to converge. Figure 10 shows a comparison of the average time per sample. Figure 11 shows the same data on a semi-log scale.

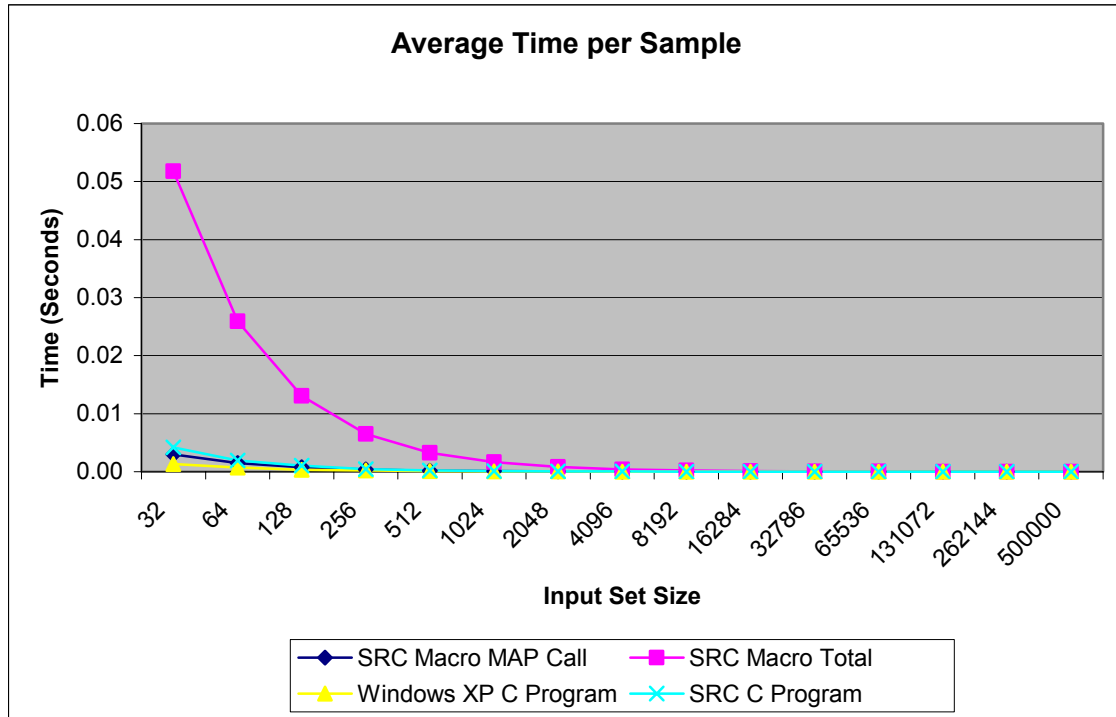


Figure 10. Comparison of Average Time per Sample

The overhead in the SRC Macro clearly dominates the graphs for smaller sample set sizes. However, for larger sample set sizes, the overhead time is amortized over the total time to be nearly insignificant. Figure 12 shows only the upper sample set size data to magnify the differences. The SRC Macro Total time is approaching the other curves and presumably would eventually meet them if the sample set size could be further increased. However, this is not possible with the current macro design due to the memory design of the SRC-6E hardware.

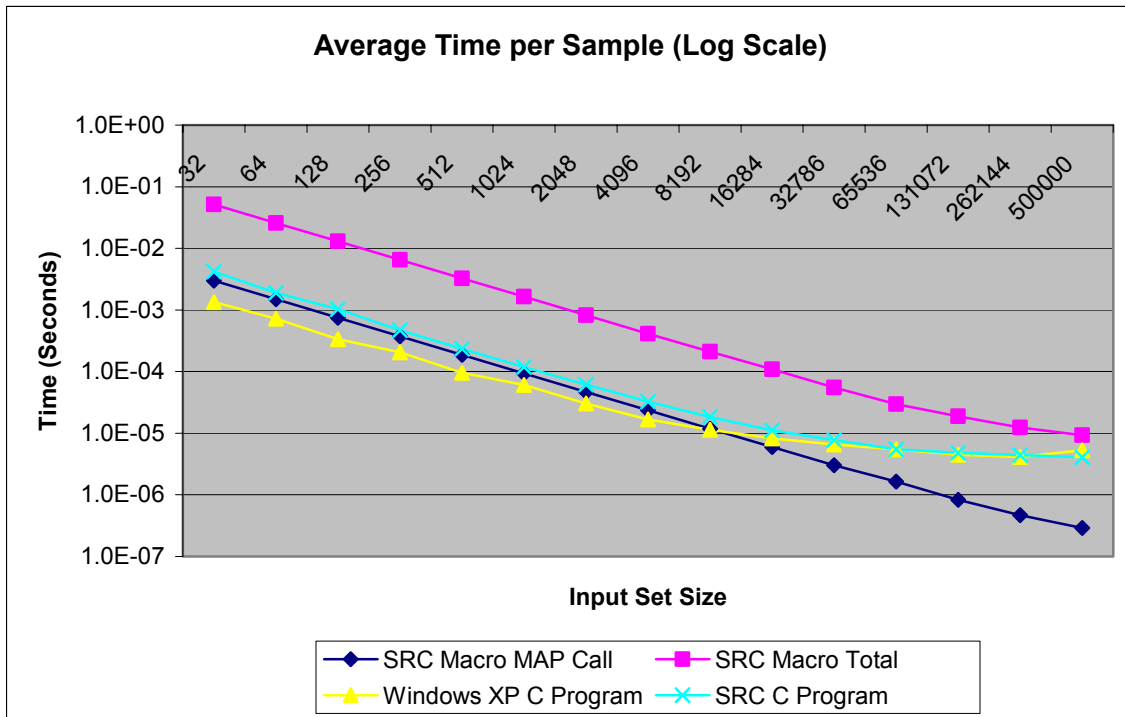


Figure 11. Semi-Log Comparison of Average Time per Sample

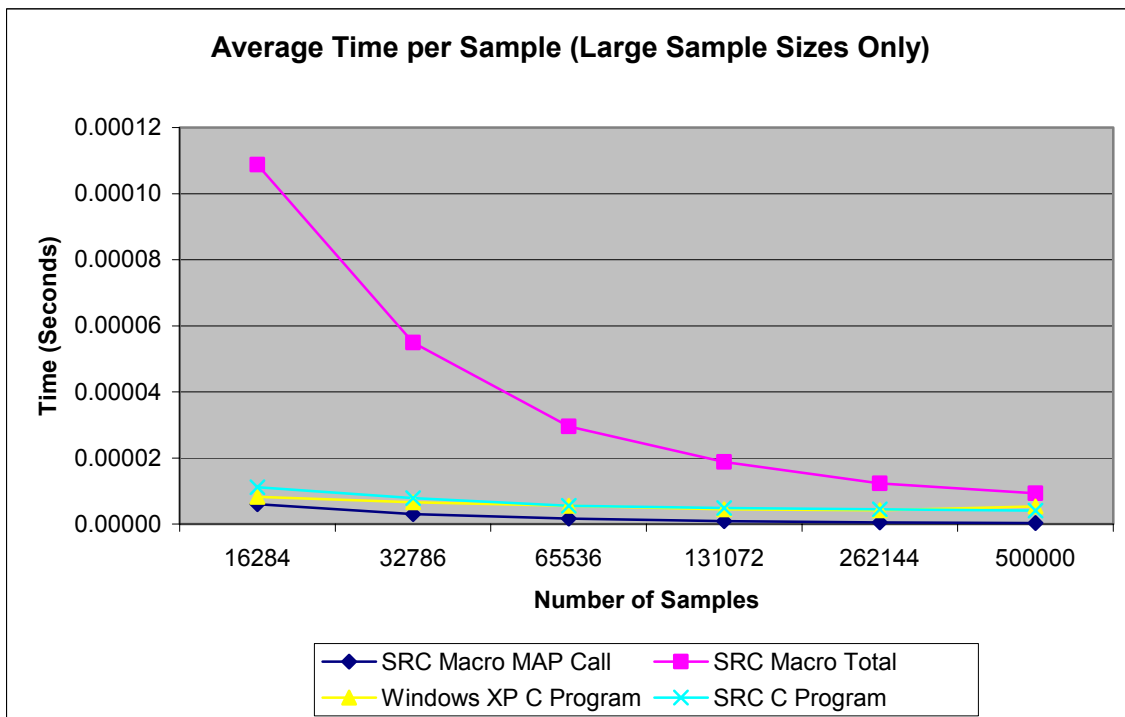


Figure 12. Comparison of Average Time per Sample

3. Conclusions

The design of the VHDL macro running on the SRC-6E suffers from excessive overhead which makes it less efficient than the C program which performs the same calculations. Due to the memory size available to the user logic on the SRC-6E, the sample set size cannot be increased large enough to make the VHDL macro run efficiently. The calculation time on the SRC user logic is extremely fast but this is irrelevant if a method cannot be developed to reduce the overhead.

The C program running on Windows is faster at low sample set sizes due to the raw processing power of the faster clocked Pentium 4. However, the slower Linux based SRC system catches up for larger sample set sizes and even appears to surpass the Pentium at the 500,000 sample set size. It appears that the Linux operating system is more efficient than Windows for this particular algorithm on the SRC-6E. However, the much greater cost of the SRC-6E does not make it a cost-effective solution for this algorithm.

This chapter discussed benchmarking the SRC-6E, including collection of data and analysis, and drew conclusions on the results. The next chapter draws conclusions on the SRC-6E, including difficulty of use and appropriateness for the chosen algorithm. Recommendations for future work are also presented.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

A. INTRODUCTION

This chapter draws conclusions on the difficulty of use of the SRC-6E, appropriateness of the chosen algorithm for application on the SRC-6E, and gives recommendations for future work.

B. DIFFICULTY OF USE

1. Necessary Skills

Programming the SRC-6E to use user-defined macros requires knowledge of high-level programming languages, hardware description languages, hardware component design, and synthesizability. Relatively few people possess all of these skills to use the system effectively without first receiving significant training. However, programming the system using only high-level languages of C or Fortran is possible which widens the potential user base to many more people. Much more research needs to be performed to determine if either method produces more effective solutions.

2. Experience Level

The SRC-6E has a relatively steep learning curve. There are a few examples in the documentation and a very small body of work in place using the system. The errors generated by the system during development are not intuitive and cannot be solved without previous experience with solving the same errors. The SRC support staff are very helpful in solving specific code problems but are not forthcoming in the reasons or methods used to resolve them. There are no development tools in place to assist novice users in programming the system. More research is required to see how much experience on the system is required to prevent and or recognize these types of errors quickly.

3. Development Time

The development time to implement solutions on this system appears to be high, primarily due to the steep learning curve and lack of development tools. This research represents approximately one year of part-time work by a single, previously inexperienced person, of which about half the time was working with the SRC-6E. It should be noted that many delays were present in the research that would not occur on a second attempt at testing the system, for example, scheduling user training and initial delivery of the system. More research must be performed to further quantify the development time and see how it improves once a group of experienced repeat users is grown. No research has yet been performed with large projects, employing multiple programmers, to see if the total project time can be reduced effectively.

C. APPROPRIATENESS OF THIS ALGORITHM

The chosen implementation of the false target radar imaging algorithm appears to be one that would benefit from a reconfigurable computer because it is pipelined and supports parallel processing. However, implementation of the design with four or less range bins has been shown to lack the necessary parallelism required to fully utilize the hardware and make it effective. Without increases in the memory size allocated for the user logic, implementation of four range bins on the SRC-6E is not an effective solution in terms of development time, processing time, or cost-effectiveness.

D. RECOMMENDATIONS FOR FUTURE WORK

1. Develop Implementation of More Range Bins.

The algorithm is not parallel enough with four or less range bins to make implementing it on the SRC-6E architec-

ture an effective solution. Expanding the interface to instantiate and deliver data to more range bins at once may show a more drastic increase in performance versus other computing methods. Rough estimates of FPGA usage show that 16 range bins should fit in the user logic area. However, rebuilding the interface to support this could be a challenge with the limited bandwidth provided by six 64-bit arrays.

2. Develop a More User-Friendly Programming Environment.

As previously discussed, the SRC-6E lacks a custom code editing environment with modern features such as real time syntax checking. Automated generation of some of the support files could also be implemented. Project wizards could be created that ask a few questions and then create the skeletons of the support files for the project. Changes to one file that affect another could be automatically corrected or at a minimum generate warnings.

3. Testing Other Applications.

The knowledge base of what types of applications do or do not work efficiently on this system is very small. Many more algorithms need to be tested on the system. Programming the same algorithm with both the high level language method and the user macro method would also provide information on which produces better results for different types of algorithms. Cost and timing comparison to modern, readily available computers should continue to be made.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

This appendix contains the C code written by Professor Douglas Fouts that was used as a standard for output correctness and as a source of timing data. Slight modifications were made to provide for timing result output and increased sample sizes. The version presented was used on both the SRC-6E and the Windows XP platforms for timing analysis with no modifications.

A. CHIP2_SIM.C

```
/* Simulate the DIS-512 chip. */
/* Compile Command */
/* cc Chip2_Sim.c -lm */
/* Range bin phase increment data must be in the file phzinc.txt. */
/* Range bin amplitude scaling data must be in the file ampscal.txt. */
/* Pulse phase samples must be in the file phzsamp.txt. */
/* Output results are put into the file IandQout.txt */
/* Global Included Files */
#include <stdio.h>
#include <math.h>
#include <time.h>
/* Global Defines */
#define rangebins 4      /* Number of range bins. */
#define phzsamps 500000  /* Maximum number of phase samples. */
/* Global Data Structures */
int phzincdat[rangebins], /* Stores phase increments for each range
bin. */
    ampscaldat[rangebins], /* Stores amplitude scaling factors for
each range bin. */
    Ipartres[phzsamps + rangebins][rangebins], /* Stores partial
results for each phase sample */
    Qpartres[phzsamps + rangebins][rangebins], /* in each
range bin. */
    sintab[32], costab[32], /* Sin and Cos lookup tables. */
    numofsamps; /* Used to count number of samples read in from
file phzsamp.txt. */
```

```

/* Read in phase increment values for each range bin, */
/* and store the results in the array phzincdat.      */
rdphzinc()
{
    /* Local Variables */
    FILE *filepnt;
    int rbcnt;
    /* Open the input file phzinc.txt. */
    if ((filepnt = fopen("phzinc.txt", "r")) == NULL)
        fprintf(stderr, "\n\nTERMINAL FAULT: File phzinc.txt not
found.\n\n");
    /* For each range bin. */
    for (rbcnt = 0; rbcnt < rangebins; rbcnt++)
    {
        fscanf(filepnt, "%x", &phzincdat[rbcnt]); /* Read in phase
increment value. */
    } /* end of for loop */
    /* Close input file. */
    fclose(filepnt);
} /* End of function rdphzinc. */

/* Read in amplitude scaling values for each range bin, */
/* and store result in array ampscaldata.                */
rdampscal()
{
    /* Local Variables */
    FILE *filepnt;
    int rbcnt, inptampdat, tstampdat;

    /* Open the input file ampscal.txt. */
    if ((filepnt = fopen("ampscal.txt", "r")) == NULL)
        fprintf(stderr, "\n\nTERMINAL FAULT: File ampscal.txt not
found.\n\n");
    /* Read in amplitude scaling values for each range bin. */
    for (rbcnt = 0; rbcnt < rangebins; rbcnt++)
    {
        fscanf(filepnt, "%x", &inptampdat);
        ampscaldata[rbcnt] = 0x00000001 & inptampdat;
    }
}

```

```

        tstampdat = 0x00000001 & (inptampdat >> 1);
    if (tstampdat == 1)
        ampscaldat[rbcnt] = ampscaldat[rbcnt] + 2;
    tstampdat = 0x00000001 & (inptampdat >> 2);
    if (tstampdat == 1)
        ampscaldat[rbcnt] = ampscaldat[rbcnt] + 3;
    tstampdat = 0x00000001 & (inptampdat >> 3);
    if (tstampdat == 1)
        ampscaldat[rbcnt] = ampscaldat[rbcnt] + 4;
}
/* Close input file. */
fclose(filepnt);
} /* End of function rdampscal. */

/* Initialize the global storage arrays. */
initarrays()
{
    /* Local Variables */
    int sampnum, rbnum;

    /* Initialize the partial result array. */
    for (sampnum = 0; sampnum < phzsamps; sampnum++)
        for (rbnum = 0; rbnum < rangebins; rbnum++)
        {
            Ipartres[sampnum + rbnum][rbnum] = 0;
            Qpartres[sampnum + rbnum][rbnum] = 0;
        }

    /* Initialize the sin table. */
    sintab[0] = 0x00000000;
    sintab[1] = 0x00000019;
    sintab[2] = 0x00000031;
    sintab[3] = 0x00000047;
    sintab[4] = 0x0000005A;
    sintab[5] = 0x0000006A;
    sintab[6] = 0x00000075;
    sintab[7] = 0x0000007D;
    sintab[8] = 0x0000007F;

```

```

sintab[9] = 0x0000007D;
sintab[10] = 0x00000075;
sintab[11] = 0x0000006A;
sintab[12] = 0x0000005A;
sintab[13] = 0x00000047;
sintab[14] = 0x00000031;
sintab[15] = 0x00000019;
sintab[16] = 0x00000000;
sintab[17] = 0xFFFFFFE7;
sintab[18] = 0xFFFFFCF;
sintab[19] = 0xFFFFFB9;
sintab[20] = 0xFFFFFA6;
sintab[21] = 0xFFFFF96;
sintab[22] = 0xFFFFF8B;
sintab[23] = 0xFFFFF83;
sintab[24] = 0xFFFFF81;
sintab[25] = 0xFFFFF83;
sintab[26] = 0xFFFFF8B;
sintab[27] = 0xFFFFF96;
sintab[28] = 0xFFFFFA6;
sintab[29] = 0xFFFFFB9;
sintab[30] = 0xFFFFFCF;
sintab[31] = 0xFFFFFE7;

/* Initialize the cos table. */
costab[0] = 0x0000007F;
costab[1] = 0x0000007D;
costab[2] = 0x00000075;
costab[3] = 0x0000006A;
costab[4] = 0x0000005A;
costab[5] = 0x00000047;
costab[6] = 0x00000031;
costab[7] = 0x00000019;
costab[8] = 0x00000000;
costab[9] = 0xFFFFFE7;
costab[10] = 0xFFFFFCF;
costab[11] = 0xFFFFFB9;
costab[12] = 0xFFFFFA6;

```

```

    costab[13] = 0xFFFFFFFF96;
    costab[14] = 0xFFFFFFFF8B;
    costab[15] = 0xFFFFFFFF83;
    costab[16] = 0xFFFFFFFF81;
    costab[17] = 0xFFFFFFFF83;
    costab[18] = 0xFFFFFFFF8B;
    costab[19] = 0xFFFFFFFF96;
    costab[20] = 0xFFFFFFFFA6;
    costab[21] = 0xFFFFFFFFB9;
    costab[22] = 0xFFFFFFFFCF;
    costab[23] = 0xFFFFFFFFE7;
    costab[24] = 0x00000000;
    costab[25] = 0x00000019;
    costab[26] = 0x00000031;
    costab[27] = 0x00000047;
    costab[28] = 0x0000005A;
    costab[29] = 0x0000006A;
    costab[30] = 0x00000075;
    costab[31] = 0x0000007D;
}      /* End of function initarrays. */

/* Read in pulse phase samples and calculate partial */
/* results for each range bin and store result in      */
/* the arrays Ipartres and Qpartres.                    */
rdphzsamp()
{
    /* Local Variables */
    FILE *filepnt;
    int phzdat, phzaddout, ILUTOut, QLUTOut, IGainOut, QGainOut, rbcnt;

    /* Open the input file phzsamp.txt. */
    if ((filepnt = fopen("phzsamp.txt", "r")) == NULL)
        fprintf(stderr, "\n\nTERMINAL FAULT:  File phzsamp.txt not
found.\n\n");

    /* Process each phase sample in the file phzsamp.txt. */
    numofsamps = 0;
    while (fscanf(filepnt, "%x", &phzdat) != EOF)
    {

```

```

    /* Process the new phase sample in each range bin and store the
    result. */
    for (rbcnt = 0; rbcnt < rangebins; rbcnt++)
    {
        /* Increment the phase. */
        phzaddout = phzdat + phzinmdat[rbcnt];
        phzaddout = phzaddout & 0x0000001F;
        /* Calculate I for each range bin and store the result. */
        ILUTOut = costab[phzaddout];
        IGainOut = ILUTOut << ampscaldata[rbcnt];
        if (IGainOut >= 0)
            IGainOut = IGainOut >> 5;
        else
            IGainOut = (IGainOut >> 5) | 0xFFFFE000;
        IGainOut = IGainOut & 0x0000FFFF;
        Ipartres[numofsamps + rbcnt][rbcnt] = IGainOut;
        /* Calculate Q for each range bin and store the result. */
        QLUTOut = sintab[phzaddout];
        QGainOut = QLUTOut << ampscaldata[rbcnt];
        if (QGainOut >= 0)
            QGainOut = QGainOut >> 5;
        else
            QGainOut = (QGainOut >> 5) | 0xFFFFE000;
        QGainOut = QGainOut & 0x0000FFFF;
        Qpartres[numofsamps + rbcnt][rbcnt] = QGainOut;
    }
    /* Increment the number of phase samples counter. */
    numofsamps++;
} /* End of outside while loop. */
/* Close input file. */
fclose(filepnt);
} /* End of function rdphzsamp. */

/* Sum partial results in the array partres and write */
/* final sums to the output file IandQout.txt. */
sumpartres()
{
    /* Local Variables */

```

```

FILE *filepnt;

int sampnum, rbnum, finIout, finQout, IOF, QOF, signifA, signifB,
signofsum;

/* Open output file for writing. */
if ((filepnt = fopen("IandQout.txt", "w")) == NULL)
    fprintf(stderr, "\n\nTERMINAL FAULT: File IandQout.txt cannot be
written.\n\n");

/* put headers in output file */
fprintf(filepnt, " I_OF_Out      Iout      Q_OF_Out      Qout\n");
fprintf(filepnt, " -----      -----      -----      ----- \n\n");
/* for all phase samples that were read in */
for (sampnum = 0; sampnum < (numofsamps + rangebins - 1); sampnum++)
{
    finIout = finQout = IOF = QOF = 0;      /* initialize final result
*/
    rbnum = rangebins - 1;
    while (rbnum >= 0)
    {
        signifA = (finIout >> 15) & 0x00000001;
        signifB = (Ipartres[sampnum][rbnum] >> 15) & 0x00000001;
        finIout = (finIout + Ipartres[sampnum][rbnum]) & 0x0000FFFF;
        signofsum = (finIout >> 15) & 0x00000001;
        if ((signifA == 0) && (signifB == 0) && (signofsum == 1))
            IOF = 1;
        if ((signifA == 1) && (signifB == 1) && (signofsum == 0))
            IOF = 1;

        signifA = (finQout >> 15) & 0x00000001;
        signifB = (Qpartres[sampnum][rbnum] >> 15) & 0x00000001;
        finQout = (finQout + Qpartres[sampnum][rbnum]) & 0x0000FFFF;
        signofsum = (finQout >> 15) & 0x00000001;
        if ((signifA == 0) && (signifB == 0) && (signofsum == 1))
            QOF = 1;
        if ((signifA == 1) && (signifB == 1) && (signofsum == 0))
            QOF = 1;

        rbnum--;
    }
    /* Print out result to output file. */

```

```

        fprintf(filepnt, "      %d      0x%04X      %d      0x%04X\n",
IOF, finIout, QOF, finQout);
    }    /* end of outer for loop */
    /* Close output file. */
    fclose (filepnt);
}    /* End of function sumpartres. */

main()
{
    /* Local Variables */
    clock_t start, finish;
    double duration;
    FILE *filepnt;
    start=clock();

    /* Read in phase increment data for each range bin. */
    rdphzinc();
    /* Read in amplitude scaling data for each range bin. */
    rdampscal();
    /* Initialize global storage arrays. */
    initarrays();
    /* Read in pulse phase samples and calculate partial results. */
    rdphzsamp();
    /* Sum partial results and output sums. */
    sumpartres();
    finish=clock();
    duration = (double)(finish - start) / CLOCKS_PER_SEC;
    /* Open output file for writing. */
    if ((filepnt = fopen("Time.txt", "w")) == NULL)
        fprintf(stderr, "\n\nTERMINAL FAULT:  File Time.txt cannot be
written.\n\n");
    /* Print out result to output file. */
    fprintf(filepnt, "Time to complete %i samples: %2.4f seconds.\n",
numofsamps, duration);
    /* Close output file. */
    fclose (filepnt);
}    /* End of main. */

```

APPENDIX B

This appendix contains the versions of the code before they were ported to the SRC-6E.

A. D-TYPE FLIP FLOP

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity DFlipFlop is
    port (CLK, LD, RESET, D: in bit;
          Q: inout bit; Qnot: out bit := '1');
end DFlipFlop;
architecture Equations of DFlipFlop is
begin
    process (CLK, LD, RESET)
    begin
        if CLK='1' and CLK'EVENT then
            if RESET='1' then
                Q <= '0';
            elsif LD='1' then
                Q <= D;
            end if;
        end if;
    end process;
    Qnot <= not Q;
end Equations;
```

B. 5-BIT REGISTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity Register5 is
    port (CLK,LD,RESET: in bit; D5: in bit_vector (4 downto 0);
          Q5: inout bit_vector (4 downto 0); Q5not: out bit_vector (4
downto 0));
end Register5;
architecture Register5 of Register5 is
```

```

component DFlipFlop
    port (CLK, LD, RESET, D: in bit;
          Q: inout bit; Qnot: out bit);
end component;

begin
    DFF0: DFlipFlop port map (CLK, LD, RESET, D5(0), Q5(0),
                              Q5not(0));
    DFF1: DFlipFlop port map (CLK, LD, RESET, D5(1), Q5(1),
                              Q5not(1));
    DFF2: DFlipFlop port map (CLK, LD, RESET, D5(2), Q5(2),
                              Q5not(2));
    DFF3: DFlipFlop port map (CLK, LD, RESET, D5(3), Q5(3),
                              Q5not(3));
    DFF4: DFlipFlop port map (CLK, LD, RESET, D5(4), Q5(4),
                              Q5not(4));
end Register5;

```

C. 8-BIT REGISTER

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Register8 is
    port (CLK, LD, RESET: in bit; D8: in bit_vector (7 downto 0);
          Q8: inout bit_vector (7 downto 0); Q8not: out bit_vector (7
downto 0));
end Register8;

architecture Register8 of Register8 is
    component DFlipFlop
        port (CLK, LD, RESET, D: in bit;
              Q: inout bit; Qnot: out bit);
    end component;

    begin
        DFF0: DFlipFlop port map (CLK, LD, RESET, D8(0), Q8(0),
                                  Q8not(0));
        DFF1: DFlipFlop port map (CLK, LD, RESET, D8(1), Q8(1),
                                  Q8not(1));
        DFF2: DFlipFlop port map (CLK, LD, RESET, D8(2), Q8(2),
                                  Q8not(2));
        DFF3: DFlipFlop port map (CLK, LD, RESET, D8(3), Q8(3),
                                  Q8not(3));
        DFF4: DFlipFlop port map (CLK, LD, RESET, D8(4), Q8(4),
                                  Q8not(4));
    end
end Register8;

```

```

        DFF5: DFlipFlop port map (CLK, LD, RESET, D8(5), Q8(5),
Q8not(5));
        DFF6: DFlipFlop port map (CLK, LD, RESET, D8(6), Q8(6),
Q8not(6));
        DFF7: DFlipFlop port map (CLK, LD, RESET, D8(7), Q8(7),
Q8not(7));
end Register8;

```

D. 13-BIT REGISTER

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity Register13 is
    port (CLK, LD, RESET: in bit; D13: in bit_vector (12 downto 0);
        Q13: inout bit_vector (12 downto 0); Q13not: out bit_vector (12
downto 0));
end Register13;
architecture Register13 of Register13 is
    component DFlipFlop
        port (CLK, LD, RESET, D: in bit;
            Q: inout bit; Qnot: out bit);
    end component;
begin
    DFF0: DFlipFlop port map (CLK, LD, RESET, D13(0), Q13(0),
Q13not(0));
    DFF1: DFlipFlop port map (CLK, LD, RESET, D13(1), Q13(1),
Q13not(1));
    DFF2: DFlipFlop port map (CLK, LD, RESET, D13(2), Q13(2),
Q13not(2));
    DFF3: DFlipFlop port map (CLK, LD, RESET, D13(3), Q13(3),
Q13not(3));
    DFF4: DFlipFlop port map (CLK, LD, RESET, D13(4), Q13(4),
Q13not(4));
    DFF5: DFlipFlop port map (CLK, LD, RESET, D13(5), Q13(5),
Q13not(5));
    DFF6: DFlipFlop port map (CLK, LD, RESET, D13(6), Q13(6),
Q13not(6));
    DFF7: DFlipFlop port map (CLK, LD, RESET, D13(7), Q13(7),
Q13not(7));
    DFF8: DFlipFlop port map (CLK, LD, RESET, D13(8), Q13(8),
Q13not(8));
    DFF9: DFlipFlop port map (CLK, LD, RESET, D13(9), Q13(9),
Q13not(9));

```

```

        DFF10: DFlipFlop port map (CLK, LD, RESET, D13(10), Q13(10),
Q13not(10));

        DFF11: DFlipFlop port map (CLK, LD, RESET, D13(11), Q13(11),
Q13not(11));

        DFF12: DFlipFlop port map (CLK, LD, RESET, D13(12), Q13(12),
Q13not(12));

end Register13;

```

E. 17-BIT REGISTER

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Register17 is
    port (CLK, LD, RESET: in bit; D17: in bit_vector (16 downto 0);
        Q17: inout bit_vector (16 downto 0); Q17not: out bit_vector (16
downto 0));
end Register17;

architecture Register17 of Register17 is
    component DFlipFlop
        port (CLK, LD, RESET,D: in bit;
            Q: inout bit; Qnot: out bit);
    end component;

begin
    DFF0: DFlipFlop port map (CLK, LD, RESET, D17(0), Q17(0),
Q17not(0));

    DFF1: DFlipFlop port map (CLK, LD, RESET, D17(1), Q17(1),
Q17not(1));

    DFF2: DFlipFlop port map (CLK, LD, RESET, D17(2), Q17(2),
Q17not(2));

    DFF3: DFlipFlop port map (CLK, LD, RESET, D17(3), Q17(3),
Q17not(3));

    DFF4: DFlipFlop port map (CLK, LD, RESET, D17(4), Q17(4),
Q17not(4));

    DFF5: DFlipFlop port map (CLK, LD, RESET, D17(5), Q17(5),
Q17not(5));

    DFF6: DFlipFlop port map (CLK, LD, RESET, D17(6), Q17(6),
Q17not(6));

    DFF7: DFlipFlop port map (CLK, LD, RESET, D17(7), Q17(7),
Q17not(7));

    DFF8: DFlipFlop port map (CLK, LD, RESET, D17(8), Q17(8),
Q17not(8));

    DFF9: DFlipFlop port map (CLK, LD, RESET, D17(9), Q17(9),
Q17not(9));

```

```

        DFF10: DFlipFlop port map (CLK, LD, RESET, D17(10), Q17(10),
Q17not(10));

        DFF11: DFlipFlop port map (CLK, LD, RESET, D17(11), Q17(11),
Q17not(11));

        DFF12: DFlipFlop port map (CLK, LD, RESET, D17(12), Q17(12),
Q17not(12));

        DFF13: DFlipFlop port map (CLK, LD, RESET, D17(13), Q17(13),
Q17not(13));

        DFF14: DFlipFlop port map (CLK, LD, RESET, D17(14), Q17(14),
Q17not(14));

        DFF15: DFlipFlop port map (CLK, LD, RESET, D17(15), Q17(15),
Q17not(15));

        DFF16: DFlipFlop port map (CLK, LD, RESET, D17(16), Q17(16),
Q17not(16));

end Register17;

```

F. FULL ADDER

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity FullAdder is
    port (X, Y, Cin: in bit;
        Cout, Sum: out bit);
end FullAdder;
architecture Equations of FullAdder is
begin
    Sum <= X xor Y xor Cin;
    Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end Equations;

```

G. FULL ADDER WITH OVERFLOW SIGNAL

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity FullAdderOV is
    port (Ci, Cout, OVin: in bit;
        Co, OVout: out bit);
end FullAdderOV;
architecture Equations of FullAdderOV is
begin
    Co <= Cout;

```

```

        OVout <= OVin or (Ci xor Cout);
end Equations;

```

H. 5-BIT ADDER

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity Adder5 is
    port (A, B: in bit_vector(4 downto 0); Ci: in bit;
          S: out bit_vector(4 downto 0); Co: out bit);
end Adder5;
architecture Adder5 of Adder5 is
    component FullAdder
        port (X, Y, Cin: in bit;
              Cout, Sum: out bit);
    end component;
    signal C: bit_vector(4 downto 1);
begin
    FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map (A(3), B(3), C(3), C(4), S(3));
    FA4: FullAdder port map (A(4), B(4), C(4), Co, S(4));
end Adder5;

```

I. 16-BIT ADDER WITH OVERFLOW SIGNAL

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity Adder16 is
    port (A, B: in bit_vector(15 downto 0); Ci, OVin: in bit;
          S: out bit_vector(16 downto 0); Co: out bit);
end Adder16; --bit 16 of S is overflow
architecture Adder16 of Adder16 is
    component FullAdder
        port (X, Y, Cin: in bit;
              Cout, Sum: out bit);
    end component;
    component FullAdderOV
        port (Ci, Cout, OVin: in bit;

```

```

        Co, OVout: out bit);
end component;
signal C: bit_vector(16 downto 1);
begin
    FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map (A(3), B(3), C(3), C(4), S(3));
    FA4: FullAdder port map (A(4), B(4), C(4), C(5), S(4));
    FA5: FullAdder port map (A(5), B(5), C(5), C(6), S(5));
    FA6: FullAdder port map (A(6), B(6), C(6), C(7), S(6));
    FA7: FullAdder port map (A(7), B(7), C(7), C(8), S(7));
    FA8: FullAdder port map (A(8), B(8), C(8), C(9), S(8));
    FA9: FullAdder port map (A(9), B(9), C(9), C(10), S(9));
    FA10: FullAdder port map (A(10), B(10), C(10), C(11), S(10));
    FA11: FullAdder port map (A(11), B(11), C(11), C(12), S(11));
    FA12: FullAdder port map (A(12), B(12), C(12), C(13), S(12));
    FA13: FullAdder port map (A(13), B(13), C(13), C(14), S(13));
    FA14: FullAdder port map (A(14), B(14), C(14), C(15), S(14));
    FA15: FullAdder port map (A(15), B(15), C(15), C(16), S(15));
    FAOV: FullAdderOV port map (C(15), C(16), OVin, Co, S(16));
end Adder16;

```

J. LUT

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use bit_pack.all;
entity ROMLUT is
port (SIN, COS:out bit_vector(8 downto 1);
      FIVEBITS:in bit_vector(5 downto 1));
end ROMLUT;
architecture ROMLUT of ROMLUT is
type ROMLUT is array (0 to 31) of bit_vector(15 downto 0);
constant FSM_ROMLUT: ROMLUT := -- 8 bits of sine and 8 bits of cosine
("0000000001111111","0001100101111101","0011000101110101","010001110110
1010","0101101001011010","0110101001000111","0111010100110001","0111110
100011001","0111111100000000","0111110111100111","0111010111001111","01
10101010111001","0101101010100110","0100011110010110","0011000110001011
","0001100110000011","0000000010000001","1110011110000011","11001111100

```

```

01011","1011100110010110","1010011010100110","1001011010111001","100010
1111001111","1000001111100111","1000000100000000","1000001100011001","1
000101100110001","1001011001000111","1010011001011010","101110010110101
0","1100111101110101","1110011101111101");

```

```

begin
    process      (FIVEBITS)
        variable ROMLUTValue: bit_vector(15 downto 0);
    begin
        ROMLUTValue:= FSM_ROMLUT(vec2int(FIVEBITS));
        SIN <= ROMLUTValue(15 downto 8);
        COS <= ROMLUTValue(7 downto 0);
    end process;
end ROMLUT;

```

K. CONTROL LOGIC BLOCK

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity ControlLogic is
    port (ODVin, URB, PSVin, CLK, OPER: in bit;
          CLR13, CLR17: out bit := '1'; ODVout, PSVout: out bit);
end ControlLogic;
architecture ControlLogic of ControlLogic is
    component DFlipFlop
        port (CLK, LD, RESET, D: in bit;
              Q: inout bit; Qnot: out bit);
    end component;
    signal
    RESET,D1,Q1,Q1Not,D2,Q2,Q2Not,D3,Q3,Q3Not,D4,Q4,Q4Not,PSVD,PSVQ,PSVQNot
    :bit;
begin
    RESET <= '0';
    PSVFF: DFlipFlop port map (CLK, OPER, RESET, PSVD, PSVQ,
    PSVQNot);
    DFF1: DFlipFlop port map(CLK, OPER, RESET, D1, Q1, Q1Not);
    DFF2: DFlipFlop port map(CLK, OPER, RESET, D2, Q2, Q2Not);
    DFF3: DFlipFlop port map(CLK, OPER, RESET, D3, Q3, Q3Not);
    DFF4: DFlipFlop port map(CLK, OPER, RESET, D4, Q4, Q4Not);
    process (URB, ODVin, PSVin)
    begin
        PSVD <= PSVin;

```

```

        D1 <= URB and PSVQ;
        D2 <= Q1;
        D3 <= ODVin or Q2;
        D4 <= Q3;
        CLR13 <= Q2Not;
        CLR17 <= Q3Not;
        PSVout <= PSVQ;
        ODVout <= Q4;
    end process;
end ControlLogic;

```

L. SHIFTER

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use bit_pack.all;
entity GainShifter is
    port (Control:in bit_vector(4 downto 1); Data: in bit_vector(8
downto 1));
        Output: out bit_vector(13 downto 1));
end GainShifter;
architecture GainShifter of GainShifter is
begin
    process      (Control, Data)
        variable C, shift, resolution, DataStop, OutStart, Out-
Stop:integer;
        variable Ones:bit_vector(13 downto 1) := "1111111111111";
    begin
        C := vec2int(Control);
        case C is
            when 0 to 2 => shift := C;
                resolution := C+3;
            when 3 to 4 => shift := 3;
                resolution :=6;
            when 5 to 7 => shift := C-1;
                if C=5 then resolution :=7;
                else resolution :=8;
                end if;
            when 8 to 10 => shift := C-4;

```

```

        if C=8 then resolution :=7;
        else resolution :=8;
        end if;
        when 11 to 12 => shift := 7;
        resolution :=8;
        when 13 to 15 => shift := C-5;
        resolution :=8;
        when others => -- summon blue screen of death
    end case;
    DataStop:=9-resolution;
    OutStart:=3+shift;
    OutStop:=Outstart-resolution+1;
    Output <= "00000000000000";
    Output(OutStart downto OutStop) <= Data (8 downto DataS-
top);
    if Data(8)='1' then      --need to preserve the sign bit
here
        Output(13 downto resolution) <= Ones(13 downto reso-
lution);
    end if;
end process;
end GainShifter;

```

M. ONE RANGE BIN

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity FakeRadarChip is
    port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1); Gain: in
bit_vector (4 downto 1);
        BinSelect: in bit_vector (9 downto 1); CLK, ODVin, URB, PSVin,
OPER, PRB, UNP: in bit;
        OtherBinDataSIN, OtherBinDataCOS: in bit_vector (17 downto 1);
        Q, I: out bit_vector (17 downto 1); ODVout, PSVout: out bit;
DRFM: out bit_vector (5 downto 1));
end FakeRadarChip;
architecture FakeRadarChip of FakeRadarChip is
    component Register5 is
        port (CLK, LD, RESET: in bit; D5: in bit_vector (4 downto 0);
            Q5: inout bit_vector (4 downto 0); Q5not: out bit_vector (4
downto 0));

```

```

end component;

component Register8 is
    port (CLK, LD, RESET: in bit; D8: in bit_vector (7 downto 0);
          Q8: inout bit_vector (7 downto 0); Q8not: out bit_vector (7
downto 0));
end component;

component Register13 is
    port (CLK, LD, RESET: in bit; D13: in bit_vector (12 downto 0);
          Q13: inout bit_vector (12 downto 0); Q13not: out bit_vector (12
downto 0));
end component;

component Register17 is
    port (CLK, LD, RESET: in bit; D17: in bit_vector (16 downto 0);
          Q17: inout bit_vector (16 downto 0); Q17not: out bit_vector (16
downto 0));
end component;

component Adder5 is
    port (A, B: in bit_vector(4 downto 0); Ci: in bit;
          S: out bit_vector(4 downto 0); Co: out bit);
end component;

component Adder16 is
    port (A, B: in bit_vector(15 downto 0); Ci, OVin: in bit;
          S: out bit_vector(16 downto 0); Co: out bit);
end component;

component ROMLUT is
port (SIN, COS:out bit_vector(1 to 8);
      FIVEBITS:in bit_vector(1 to 5));
end component;

component GainShifter is
    port (Control:in bit_vector(4 downto 1); Data: in bit_vector(8
downto 1);
          Output: out bit_vector(13 downto 1));
end component;

component ControlLogic is
    port (ODVin, URB, PSVin, CLK, OPER: in bit;
          CLR13, CLR17, ODVout, PSVout: out bit);
end component;

```

```

signal
QOutReg1,QNotOutReg1,QOutReg2,QNotOutReg2,QOutReg3,QNotOutReg3,QOutReg4
,QNotOutReg4,

    QOutReg5,QNotOutReg5,QOutReg6,QNotOutReg6,OutAdd1: bit_vector (5
downto 1);

signal QOutReg7,QNotOutReg7,QOutReg8,QNotOutReg8, LUTSIN, LUTCOS:
bit_vector (8 downto 1);

signal QOutReg9,QNotOutReg9,QOutReg10,QNotOutReg10, OutShiftSIN, Out-
ShiftCOS: bit_vector (13 downto 1);

signal
QOutReg11,QNotOutReg11,QOutReg12,QNotOutReg12,QOutReg13,QNotOutReg13,QO
utReg14,QNotOutReg14,

OutAdd2, OutAdd3: bit_vector (17 downto 1);

signal InputAdder2, InputAdder3: bit_vector (16 downto 1);

signal LD, CLR5, CLR8, CLR13, CLR17, Ci, Co1, Co2, Co3: bit;

signal InReg5: bit_vector (5 downto 1);

begin

    CLR5 <= '0';
    CLR8 <= '0';
    Ci <= '0';
    LD <= OPER;

    InReg5(4 downto 1) <= Gain (4 downto 1);
    InReg5(5) <= URB;

    Reg1: Register5 port map(CLK, LD, CLR5, PhaseInc(5 downto 1),
QOutReg1(5 downto 1), QNotOutReg1(5 downto 1));

    Reg2: Register5 port map(CLK, LD, CLR5, QOutReg1(5 downto 1),
QOutReg2(5 downto 1), QNotOutReg2(5 downto 1));

    Reg3: Register5 port map(CLK, LD, CLR5, PhaseSamp(5 downto 1),
QOutReg3(5 downto 1), QNotOutReg3(5 downto 1));

    Add1: Adder5 port map (QOutReg2,QOutReg3, Ci, OutAdd1(5 downto
1), Co1);

    Reg4: Register5 port map(CLK, LD, CLR5, OutAdd1(5 downto 1),
QOutReg4(5 downto 1), QNotOutReg4(5 downto 1));

    LUT: ROMLUT port map (LUTSIN(8 downto 1),LUTCOS(8 downto
1),QOutReg4(5 downto 1));

    Reg5: Register5 port map(CLK, LD, CLR5, InReg5(5 downto 1),
QOutReg5(5 downto 1), QNotOutReg5(5 downto 1));

    Reg6: Register5 port map(CLK, LD, CLR5, QOutReg5(5 downto 1),
QOutReg6(5 downto 1), QNotOutReg6(5 downto 1));

    Reg7: Register8 port map(CLK, LD, CLR8, LUTSIN(8 downto 1),
QOutReg7(8 downto 1), QNotOutReg7(8 downto 1));

    Reg8: Register8 port map(CLK, LD, CLR8, LUTCOS(8 downto 1),
QOutReg8(8 downto 1), QNotOutReg8(8 downto 1));

```

```

    Shift1: GainShifter port map (QOutReg6(4 downto 1),QOutReg7(8
downto 1),OutShiftSIN(13 downto 1));

    Shift2: GainShifter port map (QOutReg6(4 downto 1),QOutReg8(8
downto 1),OutShiftCOS(13 downto 1));

    Reg9: Register13 port map(CLK, LD, CLR13, OutShiftSIN(13 downto
1), QOutReg9(13 downto 1), QNotOutReg9(13 downto 1));

    Reg10: Register13 port map(CLK, LD, CLR13, OutShiftCOS(13 downto
1), QOutReg10(13 downto 1), QNotOutReg10(13 downto 1));

    Reg11: Register17 port map(CLK, LD, '0', OtherBinDataSIN(17
downto 1), QOutReg11(17 downto 1), QNotOutReg11(17 downto 1));

    Reg12: Register17 port map(CLK, LD, '0', OtherBinDataCOS(17
downto 1), QOutReg12(17 downto 1), QNotOutReg12(17 downto 1));

    Add2: Adder16 port map (InputAdder2, QOutReg11(16 downto 1), Ci,
QOutReg11(17), OutAdd2(17 downto 1),Co2);

    Add3: Adder16 port map (InputAdder3, QOutReg12(16 downto 1), Ci,
QOutReg12(17), OutAdd3(17 downto 1),Co3);

    Reg13: Register17 port map(CLK, LD, CLR17, OutAdd2(17 downto 1),
QOutReg13(17 downto 1), QNotOutReg13(17 downto 1));

    Reg14: Register17 port map(CLK, LD, CLR17, OutAdd3(17 downto 1),
QOutReg14(17 downto 1), QNotOutReg14(17 downto 1));

    Control: ControlLogic port map (ODVin, URB, PSVin, CLK, OPER,
CLR13, CLR17, ODVout, PSVout);

    InputAdder2(13 downto 1) <= QOutReg9(13 downto 1);
    InputAdder2(14) <= QOutReg9(13);
    InputAdder2(15) <= QOutReg9(13);
    InputAdder2(16) <= QOutReg9(13);
    InputAdder3(13 downto 1) <= QOutReg10(13 downto 1);
    InputAdder3(14) <= QOutReg10(13);
    InputAdder3(15) <= QOutReg10(13);
    InputAdder3(16) <= QOutReg10(13);
    DRFM(5 downto 1) <= QOutReg3(5 downto 1);
    Q <= QOutReg13;
    I <= QOutReg14;
end FakeRadarChip;

```

N. TWO RANGE BINS

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity TwoBins is
    port (PhaseSamp, PhaseInc0, PhaseInc1: in bit_vector (5 downto
1));

```

```

    Gain0, Gain1: in bit_vector (4 downto 1); BinSelect0, BinSelect1:
in bit_vector (9 downto 1);

    CLK, ODVin, URB0, URB1, PSVin, OPER0, OPER1, PRB0, PRB1, UNP0,
UNP1: in bit;

    OtherBinDataSIN, OtherBinDataCOS: in bit_vector (17 downto 1);

    Q, I: out bit_vector (17 downto 1); Q1, I1: inout bit_vector (17
downto 1); ODVout0, ODVout1, PSVout0, PSVout1: inout bit; CLR13out0,
CLR13out1, CLR17out0, CLR17out1: out bit;

    DRFM0, DRFM1: inout bit_vector (5 downto 1));
end TwoBins;

architecture TwoBins of TwoBins is
component FakeRadarChip is
    port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1); Gain: in
bit_vector (4 downto 1);

        BinSelect: in bit_vector (9 downto 1); CLK, ODVin, URB, PSVin,
OPER, PRB, UNP: in bit;

        OtherBinDataSIN, OtherBinDataCOS: in bit_vector (17 downto 1);

        Q, I: out bit_vector (17 downto 1); ODVout, PSVout, CLR13out,
CLR17out: out bit; DRFM: out bit_vector (5 downto 1));
end component;

begin -- BIN0 is the primary output

BIN0: FakeRadarChip port map (DRFM1, PhaseInc0, Gain0, BinSelect0, CLK,
ODVout1, URB0, PSVout1, OPER0,

    PRB0, UNP0, Q1, I1, Q, I, ODVout0, PSVout0, CLR13out0,
CLR17out0, DRFM0);

BIN1: FakeRadarChip port map (PhaseSamp, PhaseInc1, Gain1, BinSelect1,
CLK, ODVin, URB1, PSVin, OPER1,

    PRB1, UNP1, OtherBinDataSIN, OtherBinDataCOS, Q1, I1, OD-
Vout1, PSVout1, CLR13out1, CLR17out1, DRFM1);
end TwoBins;

```

O. FOUR RANGE BINS

```

library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity FourBins is
    port (PhaseSamp, PhaseInc0, PhaseInc1, PhaseInc2, PhaseInc3: in
bit_vector (5 downto 1);

        Gain0, Gain1, Gain2, Gain3: in bit_vector (4 downto 1);

        CLK, ODVin, PSVin: in bit; OtherBinDataSIN, OtherBinDataCOS: in
bit_vector (17 downto 1);

        Q, I: out bit_vector (17 downto 1); ODVout, PSVout: out bit;

        DRFMout: out bit_vector (5 downto 1));

```

```

end FourBins;

architecture FourBins of FourBins is
component FakeRadarChip is
    port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1); Gain: in
bit_vector (4 downto 1);

        BinSelect: in bit_vector (9 downto 1); CLK, ODVin, URB, PSVin,
OPER, PRB, UNP: in bit;

        OtherBinDataSIN, OtherBinDataCOS: in bit_vector (17 downto 1);

        Q, I: out bit_vector (17 downto 1); ODVout, PSVout: out bit;
DRFM: out bit_vector (5 downto 1));
end component;

signal Q1,I1,Q2,I2,Q3,I3: bit_vector (17 downto 1);
signal DRFM0, DRFM1,DRFM2,DRFM3: bit_vector (5 downto 1);

signal ODVout0, ODVout1, ODVout2, ODVout3,
PSVout0,PSVout1,PSVout2,PSVout3:bit;

begin -- BIN0 is the primary output
BIN0: FakeRadarChip port map (DRFM1, PhaseInc0, Gain0, "000000000",
CLK, ODVout1, '1', PSVout1, '1',
    '1', '1',    Q1, I1, Q, I, ODVout0, PSVout0, DRFM0);

BIN1: FakeRadarChip port map (DRFM2, PhaseInc1, Gain1, "000000000",
CLK, ODVout2, '1', PSVout2, '1',
    '1', '1',    Q2, I2, Q1, I1, ODVout1, PSVout1, DRFM1);

BIN2: FakeRadarChip port map (DRFM3, PhaseInc2, Gain2, "000000000",
CLK, ODVout3, '1', PSVout3, '1',
    '1', '1',    Q3, I3, Q2, I2, ODVout2, PSVout2, DRFM2);

BIN3: FakeRadarChip port map (PhaseSamp, PhaseInc3, Gain3, "000000000",
CLK, ODVin, '1', PSVin, '1',
    '1', '1', OtherBinDataSIN, OtherBinDataCOS, Q3, I3, OD-
Vout3, PSVout3, DRFM3);

ODVout<=ODVout0;
PSVout<=ODVout0;
DRFMout<=DRFM0;

end FourBins;

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

This appendix contains the final version of the VHDL code that was implemented on the SRC-6E and the support files required to compile and execute it.

A. MACRO VHDL FILE

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity DFlipFlop is
    port (CLK, LD, RESET, D: in bit;
          Q: inout bit; Qnot: out bit := '1');
end DFlipFlop;

architecture Equations of DFlipFlop is
begin
    process (CLK, LD, RESET)
    begin
        if CLK='1' and CLK'EVENT then
            if RESET='1' then
                Q <= '0';
            elsif LD='1' then
                Q <= D;
            end if;
        end if;
    end process;
    Qnot <= not Q;
end Equations;

entity Register5 is
    port (CLK,LD,RESET: in bit; D5: in bit_vector (4 downto 0);
          Q5: inout bit_vector (4 downto 0); Q5not: out bit_vector (4
downto 0));
end Register5;

architecture Register5 of Register5 is
```

```

component DFlipFlop
    port (CLK, LD, RESET, D: in bit;
          Q: inout bit; Qnot: out bit);
end component;

begin
    DFF0: DFlipFlop port map (CLK, LD, RESET, D5(0), Q5(0),
                              Q5not(0));
    DFF1: DFlipFlop port map (CLK, LD, RESET, D5(1), Q5(1),
                              Q5not(1));
    DFF2: DFlipFlop port map (CLK, LD, RESET, D5(2), Q5(2),
                              Q5not(2));
    DFF3: DFlipFlop port map (CLK, LD, RESET, D5(3), Q5(3),
                              Q5not(3));
    DFF4: DFlipFlop port map (CLK, LD, RESET, D5(4), Q5(4),
                              Q5not(4));
end Register5;

entity Register8 is
    port (CLK, LD, RESET: in bit; D8: in bit_vector (7 downto 0);
          Q8: inout bit_vector (7 downto 0); Q8not: out bit_vector (7
downto 0));
end Register8;

architecture Register8 of Register8 is
    component DFlipFlop
        port (CLK, LD, RESET, D: in bit;
              Q: inout bit; Qnot: out bit);
    end component;

    begin
        DFF0: DFlipFlop port map (CLK, LD, RESET, D8(0), Q8(0),
                                  Q8not(0));
        DFF1: DFlipFlop port map (CLK, LD, RESET, D8(1), Q8(1),
                                  Q8not(1));
        DFF2: DFlipFlop port map (CLK, LD, RESET, D8(2), Q8(2),
                                  Q8not(2));
        DFF3: DFlipFlop port map (CLK, LD, RESET, D8(3), Q8(3),
                                  Q8not(3));
        DFF4: DFlipFlop port map (CLK, LD, RESET, D8(4), Q8(4),
                                  Q8not(4));
    end
end Register8;

```

```

        DFF5: DFlipFlop port map (CLK, LD, RESET, D8(5), Q8(5),
Q8not(5));

        DFF6: DFlipFlop port map (CLK, LD, RESET, D8(6), Q8(6),
Q8not(6));

        DFF7: DFlipFlop port map (CLK, LD, RESET, D8(7), Q8(7),
Q8not(7));
end Register8;

entity Register13 is
    port (CLK, LD, RESET: in bit; D13: in bit_vector (12 downto 0);
        Q13: inout bit_vector (12 downto 0); Q13not: out bit_vector (12
downto 0));
end Register13;

architecture Register13 of Register13 is
    component DFlipFlop
        port (CLK, LD, RESET, D: in bit;
            Q: inout bit; Qnot: out bit);
    end component;

begin
    DFF0: DFlipFlop port map (CLK, LD, RESET, D13(0), Q13(0),
Q13not(0));
    DFF1: DFlipFlop port map (CLK, LD, RESET, D13(1), Q13(1),
Q13not(1));
    DFF2: DFlipFlop port map (CLK, LD, RESET, D13(2), Q13(2),
Q13not(2));
    DFF3: DFlipFlop port map (CLK, LD, RESET, D13(3), Q13(3),
Q13not(3));
    DFF4: DFlipFlop port map (CLK, LD, RESET, D13(4), Q13(4),
Q13not(4));
    DFF5: DFlipFlop port map (CLK, LD, RESET, D13(5), Q13(5),
Q13not(5));
    DFF6: DFlipFlop port map (CLK, LD, RESET, D13(6), Q13(6),
Q13not(6));
    DFF7: DFlipFlop port map (CLK, LD, RESET, D13(7), Q13(7),
Q13not(7));
    DFF8: DFlipFlop port map (CLK, LD, RESET, D13(8), Q13(8),
Q13not(8));
    DFF9: DFlipFlop port map (CLK, LD, RESET, D13(9), Q13(9),
Q13not(9));

```

```

        DFF10: DFlipFlop port map (CLK, LD, RESET, D13(10), Q13(10),
Q13not(10));

        DFF11: DFlipFlop port map (CLK, LD, RESET, D13(11), Q13(11),
Q13not(11));

        DFF12: DFlipFlop port map (CLK, LD, RESET, D13(12), Q13(12),
Q13not(12));
end Register13;

entity Register17 is
    port (CLK, LD, RESET: in bit; D17: in bit_vector (16 downto 0);
        Q17: inout bit_vector (16 downto 0); Q17not: out bit_vector (16
downto 0));
end Register17;

architecture Register17 of Register17 is
    component DFlipFlop
        port (CLK, LD, RESET,D: in bit;
            Q: inout bit; Qnot: out bit);
    end component;

begin
    DFF0: DFlipFlop port map (CLK, LD, RESET, D17(0), Q17(0),
Q17not(0));
    DFF1: DFlipFlop port map (CLK, LD, RESET, D17(1), Q17(1),
Q17not(1));
    DFF2: DFlipFlop port map (CLK, LD, RESET, D17(2), Q17(2),
Q17not(2));
    DFF3: DFlipFlop port map (CLK, LD, RESET, D17(3), Q17(3),
Q17not(3));
    DFF4: DFlipFlop port map (CLK, LD, RESET, D17(4), Q17(4),
Q17not(4));
    DFF5: DFlipFlop port map (CLK, LD, RESET, D17(5), Q17(5),
Q17not(5));
    DFF6: DFlipFlop port map (CLK, LD, RESET, D17(6), Q17(6),
Q17not(6));
    DFF7: DFlipFlop port map (CLK, LD, RESET, D17(7), Q17(7),
Q17not(7));
    DFF8: DFlipFlop port map (CLK, LD, RESET, D17(8), Q17(8),
Q17not(8));
    DFF9: DFlipFlop port map (CLK, LD, RESET, D17(9), Q17(9),
Q17not(9));

```

```

        DFF10: DFlipFlop port map (CLK, LD, RESET, D17(10), Q17(10),
Q17not(10));

        DFF11: DFlipFlop port map (CLK, LD, RESET, D17(11), Q17(11),
Q17not(11));

        DFF12: DFlipFlop port map (CLK, LD, RESET, D17(12), Q17(12),
Q17not(12));

        DFF13: DFlipFlop port map (CLK, LD, RESET, D17(13), Q17(13),
Q17not(13));

        DFF14: DFlipFlop port map (CLK, LD, RESET, D17(14), Q17(14),
Q17not(14));

        DFF15: DFlipFlop port map (CLK, LD, RESET, D17(15), Q17(15),
Q17not(15));

        DFF16: DFlipFlop port map (CLK, LD, RESET, D17(16), Q17(16),
Q17not(16));
end Register17;

```

```

entity ROMLUT is
port (SIN, COS:out bit_vector(8 downto 1);
        FIVEBITS:in bit_vector(5 downto 1));
end ROMLUT;

```

architecture ROMLUT of ROMLUT is

```

signal ROMLUTValue : bit_vector(15 downto 0);

```

```

begin

```

```

with FIVEBITS Select

```

```

    ROMLUTValue <="0000000001111111" when "00000",      --0
        "0001100101111101" when "00001",      --1
        "0011000101110101" when "00010",      --2
        "0100011101101010" when "00011",      --3
        "0101101001011010" when "00100",      --4
        "0110101001000111" when "00101",      --5
        "0111010100110001" when "00110",      --6
        "0111110100011001" when "00111",      --7
        "0111111100000000" when "01000",      --8
        "0111110111100111" when "01001",      --9
        "0111010111001111" when "01010",      --A
        "0110101010111001" when "01011",      --b

```

```

        "0101101010100110" when "01100",      --C
        "0100011110010110" when "01101",      --d
        "0011000110001011" when "01110",      --E
        "0001100110000011" when "01111",      --F
        "0000000010000001" when "10000",      --10
        "1110011110000011" when "10001",      --11
        "1100111110001011" when "10010",      --12
        "1011100110010110" when "10011",      --13
        "1010011010100110" when "10100",      --14
        "1001011010111001" when "10101",      --15
        "1000101111001111" when "10110",      --16
        "1000001111100111" when "10111",      --17
        "1000000100000000" when "11000",      --18
        "1000001100011001" when "11001",      --19
        "1000101100110001" when "11010",      --1A
        "1001011001000111" when "11011",      --1b
        "1010011001011010" when "11100",      --1C
        "1011100101101010" when "11101",      --1d
        "1100111101110101" when "11110",      --1E
        "1110011101111101" when "11111",      --1F
        "0000000000000000" when others;      --Never Occurs

    SIN <= ROMLUTValue(15 downto 8);
    COS <= ROMLUTValue(7 downto 0);
end ROMLUT;

entity FullAdder is
    port (X, Y, Cin: in bit;
          Cout, Sum: out bit);
end FullAdder;

architecture Equations of FullAdder is
begin
    Sum <= X xor Y xor Cin;

    Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end Equations;

entity FullAdderOV is

```

```

        port (Ci, Cout, OVin: in bit;
              Co, OVout: out bit);
end FullAdderOV;

architecture Equations of FullAdderOV is
begin
    Co <= Cout;
    OVout <= OVin or (Ci xor Cout);
end Equations;

entity Adder5 is
    port (A, B: in bit_vector(4 downto 0); Ci: in bit;
          S: out bit_vector(4 downto 0); Co: out bit);
end Adder5;

architecture Adder5 of Adder5 is
    component FullAdder
        port (X, Y, Cin: in bit;
              Cout, Sum: out bit);
    end component;
    signal C: bit_vector(4 downto 1);
begin
    FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map (A(3), B(3), C(3), C(4), S(3));
    FA4: FullAdder port map (A(4), B(4), C(4), Co, S(4));
end Adder5;

entity CLAH4 is
    port (A, B: in bit_vector(3 downto 0); Cin: in bit; Cout: out
          bit);
end CLAH4;

architecture CLAH4 of CLAH4 is
    signal g0, g1, g2, g3, p0, p1, p2, p3: bit;
begin
    g0 <= A(0) and B(0);

```

```

    p0 <= A(0) or B(0);
    g1 <= A(1) and B(1);
    p1 <= A(1) or B(1);
    g2 <= A(2) and B(2);
    p2 <= A(2) or B(2);
    g3 <= A(3) and B(3);
    p3 <= A(3) or B(3);

    Cout <= g3 or (p3 and g2) or (p3 and p2 and g1) or (p3 and p2 and
p1 and g0) or (p3 and p2 and p1 and p0 and Cin);
end CLAH4;

```

entity CLAH8 is

```

    port (A, B: in bit_vector(7 downto 0); Cin: in bit; Cout: out
bit);
end CLAH8;

```

architecture CLAH8 of CLAH8 is

```

signal g0, g1, g2, g3, g4, g5, g6, g7, p0, p1, p2, p3, p4, p5, p6, p7:
bit;

```

begin

```

    g0 <= A(0) and B(0);
    p0 <= A(0) or B(0);
    g1 <= A(1) and B(1);
    p1 <= A(1) or B(1);
    g2 <= A(2) and B(2);
    p2 <= A(2) or B(2);
    g3 <= A(3) and B(3);
    p3 <= A(3) or B(3);
    g4 <= A(4) and B(4);
    p4 <= A(4) or B(4);
    g5 <= A(5) and B(5);
    p5 <= A(5) or B(5);
    g6 <= A(6) and B(6);
    p6 <= A(6) or B(6);
    g7 <= A(7) and B(7);
    p7 <= A(7) or B(7);

    Cout <= g7 or (p7 and g6) or (p7 and p6 and g5) or (p7 and p6 and
p5 and g4) or (p7 and p6 and p5 and p4 and g3) or

```

```

        (p7 and p6 and p5 and p4 and p3 and g2) or (p7 and p6 and p5 and
p4 and p3 and p2 and g1) or

```

```

        (p7 and p6 and p5 and p4 and p3 and p2 and p1 and g0) or (p7 and
p6 and p5 and p4 and p3 and p2 and p1 and p0 and Cin);

```

```

end CLAH8;

```

```

entity Adder16 is

```

```

    port (A, B: in bit_vector(15 downto 0); Ci, OVin: in bit;

```

```

          S: out bit_vector(16 downto 0); Co: out bit);

```

```

end Adder16; --bit 16 of S is overflow

```

```

architecture Adder16 of Adder16 is

```

```

    component CLAH4

```

```

        port (A, B: in bit_vector(3 downto 0); Cin: in bit; Cout: out
bit);

```

```

    end component;

```

```

    component CLAH8

```

```

        port (A, B: in bit_vector(7 downto 0); Cin: in bit; Cout: out
bit);

```

```

    end component;

```

```

    component FullAdder

```

```

        port (X, Y, Cin: in bit;

```

```

              Cout, Sum: out bit);

```

```

    end component;

```

```

    component FullAdderOV

```

```

        port (Ci, Cout, OVin: in bit;

```

```

              Co, OVout: out bit);

```

```

    end component;

```

```

    signal C: bit_vector(16 downto 1);

```

```

    signal dummy1, dummy2, dummy3: bit;

```

```

begin

```

```

    FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));

```

```

    FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));

```

```

    FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));

```

```

    FA3: FullAdder port map (A(3), B(3), C(3), dummy1, S(3));

```

```

    CLAH0: CLAH4 port map (A(3 downto 0), B(3 downto 0), Ci, C(4));

```

```

    FA4: FullAdder port map (A(4), B(4), C(4), C(5), S(4));

```

```

    FA5: FullAdder port map (A(5), B(5), C(5), C(6), S(5));

```

```

    FA6: FullAdder port map (A(6), B(6), C(6), C(7), S(6));
    FA7: FullAdder port map (A(7), B(7), C(7), dummy2, S(7));
    CLAH1: CLAH8 port map (A(7 downto 0), B(7 downto 0), Ci, C(8));
    FA8: FullAdder port map (A(8), B(8), C(8), C(9), S(8));
    FA9: FullAdder port map (A(9), B(9), C(9), C(10), S(9));
    FA10: FullAdder port map (A(10), B(10), C(10), C(11), S(10));
    FA11: FullAdder port map (A(11), B(11), C(11), dummy3, S(11));
    CLAH2: CLAH4 port map (A(11 downto 8), B(11 downto 8), C(8),
C(12));

    FA12: FullAdder port map (A(12), B(12), C(12), C(13), S(12));
    FA13: FullAdder port map (A(13), B(13), C(13), C(14), S(13));
    FA14: FullAdder port map (A(14), B(14), C(14), C(15), S(14));
    FA15: FullAdder port map (A(15), B(15), C(15), C(16), S(15));
    FAOV: FullAdderOV port map (C(15), C(16), OVin, Co, S(16));
end Adder16;

entity ControlLogic is
    port (ODVin, URB, PSVin, CLK, OPER: in bit;
          CLR13, CLR17: out bit := '1'; ODVout, PSVout: out bit);
end ControlLogic;

architecture ControlLogic of ControlLogic is
    component DFlipFlop
        port (CLK, LD, RESET, D: in bit;
              Q: inout bit; Qnot: out bit);
    end component;

    signal
    RESET,D1,Q1,Q1Not,D2,Q2,Q2Not,D3,Q3,Q3Not,D4,Q4,Q4Not,PSVD,PSVQ,PSVQNot
    :bit;

    begin

        RESET <= '0';

        PSVFF: DFlipFlop port map (CLK, OPER, RESET, PSVD, PSVQ,
PSVQNot);

        DFF1: DFlipFlop port map(CLK, OPER, RESET, D1, Q1, Q1Not);
        DFF2: DFlipFlop port map(CLK, OPER, RESET, D2, Q2, Q2Not);
        DFF3: DFlipFlop port map(CLK, OPER, RESET, D3, Q3, Q3Not);
        DFF4: DFlipFlop port map(CLK, OPER, RESET, D4, Q4, Q4Not);

        PSVD <= PSVin;

```

```

    D1 <= URB and PSVQ;
    D2 <= Q1;
    D3 <= ODVin or Q2;
    D4 <= Q3;
    CLR13 <= Q2Not;
    CLR17 <= Q3Not;
    PSVout <= PSVQ;
    ODVout <= Q4;

end ControlLogic;

entity GainShifter is
    port (Control:in bit_vector(4 downto 1); Data: in bit_vector(8
downto 1);
        Output: out bit_vector(13 downto 1));
end GainShifter;

architecture GainShifter of GainShifter is

begin
    process      (Control,Data)

    begin
        Output(13 downto 1) <= "00000000000000";
        case Control is
        when "0000" => Output(3 downto 1) <= Data(8 downto 6);
            if Data(8)='1' then --need to preserve the sign bit
                Output(13 downto 4) <= "1111111111";
            end if;

        when "0001" => Output(4 downto 1) <= Data(8 downto 5);
            if Data(8)='1' then --need to preserve the sign bit
                Output(13 downto 5) <= "1111111111";
            end if;

        when "0010" => Output(5 downto 1) <= Data(8 downto 4);
            if Data(8)='1' then --need to preserve the sign bit
                Output(13 downto 6) <= "1111111111";
            end if;
        end case;
    end process;
end GainShifter;

```

```

when "0011" => Output(6 downto 1) <= Data(8 downto 3);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 7) <= "1111111";
    end if;
when "0100" => Output(6 downto 1) <= Data(8 downto 3);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 7) <= "1111111";
    end if;
when "0101" => Output(7 downto 1) <= Data(8 downto 2);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 8) <= "111111";
    end if;
when "0110" => Output(8 downto 1) <= Data(8 downto 1);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 9) <= "11111";
    end if;
when "0111" => Output(9 downto 2) <= Data(8 downto 1);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 10) <= "1111";
    end if;
when "1000" => Output(7 downto 1) <= Data(8 downto 2);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 8) <= "111111";
    end if;
when "1001" => Output(8 downto 1) <= Data(8 downto 1);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 9) <= "11111";
    end if;
when "1010" => Output(9 downto 2) <= Data(8 downto 1);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 10) <= "1111";
    end if;
when "1011" => Output(10 downto 3) <= Data(8 downto 1);
    if Data(8)='1' then --need to preserve the sign bit
        Output(13 downto 11) <= "111";
    end if;
when "1100" => Output(10 downto 3) <= Data(8 downto 1);

```

```

        if Data(8)='1' then --need to preserve the sign bit
            Output(13 downto 11) <= "111";
        end if;
    when "1101" => Output(11 downto 4) <= Data(8 downto 1);
        if Data(8)='1' then --need to preserve the sign bit
            Output(13 downto 12) <= "11";
        end if;
    when "1110" => Output(12 downto 5) <= Data(8 downto 1);
        if Data(8)='1' then --need to preserve the sign bit
            Output(13) <= '1';
        end if;
    when "1111" => Output(13 downto 6) <= Data(8 downto 1);
    when others => -- summon blue screen of death
    end case;
end process;
end GainShifter;

entity OneBin is
    port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1);
        Gain: in bit_vector (4 downto 1);
        ODVin, PSVin: in bit;
        OtherBinDataSIN, OtherBinDataCOS: in bit_vector (17 downto
1);
        Q, I: out bit_vector (17 downto 1);
        ODVout, PSVout: out bit;
        DRFM: out bit_vector (5 downto 1);
        CLK: in bit);
end OneBin;

architecture OneBin of OneBin is
    component Register5 is
        port (CLK, LD, RESET: in bit; D5: in bit_vector (4 downto 0);
            Q5: inout bit_vector (4 downto 0); Q5not: out bit_vector (4
downto 0));
    end component;
    component Register8 is
        port (CLK, LD, RESET: in bit; D8: in bit_vector (7 downto 0);

```

```

        Q8: inout bit_vector (7 downto 0); Q8not: out bit_vector (7
downto 0));
end component;
component Register13 is
    port (CLK, LD, RESET: in bit; D13: in bit_vector (12 downto 0);
        Q13: inout bit_vector (12 downto 0); Q13not: out bit_vector (12
downto 0));
end component;
component Register17 is
    port (CLK, LD, RESET: in bit; D17: in bit_vector (16 downto 0);
        Q17: inout bit_vector (16 downto 0); Q17not: out bit_vector (16
downto 0));
end component;
component Adder5 is
    port (A, B: in bit_vector(4 downto 0); Ci: in bit;
        S: out bit_vector(4 downto 0); Co: out bit);
end component;
component Adder16 is
    port (A, B: in bit_vector(15 downto 0); Ci, OVin: in bit;
        S: out bit_vector(16 downto 0); Co: out bit);
end component;
component ROMLUT is
port (SIN, COS:out bit_vector(1 to 8);
        FIVEBITS:in bit_vector(1 to 5));
end component;
component GainShifter is
    port (Control:in bit_vector(4 downto 1); Data: in bit_vector(8
downto 1);
        Output: out bit_vector(13 downto 1));
end component;
component ControlLogic is
    port (ODVin, URB, PSVin, CLK, OPER: in bit;
        CLR13, CLR17, ODVout, PSVout: out bit);
end component;

signal
QOutReg1,QNotOutReg1,QOutReg2,QNotOutReg2,QOutReg3,QNotOutReg3,QOutReg4
,QNotOutReg4,
        QOutReg5,QNotOutReg5,QOutReg6,QNotOutReg6,OutAdd1: bit_vector (5
downto 1);

```

```

signal QOutReg7,QNotOutReg7,QOutReg8,QNotOutReg8, LUTSIN, LUTCOS:
bit_vector (8 downto 1);

signal QOutReg9,QNotOutReg9,QOutReg10,QNotOutReg10, OutShiftSIN, Out-
ShiftCOS: bit_vector (13 downto 1);

signal
QOutReg11,QNotOutReg11,QOutReg12,QNotOutReg12,QOutReg13,QNotOutReg13,QO
utReg14,QNotOutReg14,
OutAdd2, OutAdd3: bit_vector (17 downto 1);

signal InputAdder2, InputAdder3: bit_vector (16 downto 1);

signal OPER, URB, LD, CLR5, CLR8, CLR13, CLR17, Ci, Co1, Co2, Co3, Re-
set_Inact: bit;

signal InReg5: bit_vector (5 downto 1);

begin

    OPER <= '1';

    URB <= '1';

    CLR5 <= '0';

    CLR8 <= '0';

    Ci <= '0';

    LD <= '1';

    Reset_Inact <= '0';

    InReg5(4 downto 1) <= Gain (4 downto 1);

    InReg5(5) <= URB;

    Reg1: Register5 port map(CLK, LD, CLR5, PhaseInc(5 downto 1),
QOutReg1(5 downto 1), QNotOutReg1(5 downto 1));

    Reg2: Register5 port map(CLK, LD, CLR5, QOutReg1(5 downto 1),
QOutReg2(5 downto 1), QNotOutReg2(5 downto 1));

    Reg3: Register5 port map(CLK, LD, CLR5, PhaseSamp(5 downto 1),
QOutReg3(5 downto 1), QNotOutReg3(5 downto 1));

    Add1: Adder5 port map (QOutReg2,QOutReg3, Ci, OutAdd1(5 downto
1), Co1);

    Reg4: Register5 port map(CLK, LD, CLR5, OutAdd1(5 downto 1),
QOutReg4(5 downto 1), QNotOutReg4(5 downto 1));

    LUT: ROMLUT port map (LUTSIN(8 downto 1),LUTCOS(8 downto
1),QOutReg4(5 downto 1));

    Reg5: Register5 port map(CLK, LD, CLR5, InReg5(5 downto 1),
QOutReg5(5 downto 1), QNotOutReg5(5 downto 1));

    Reg6: Register5 port map(CLK, LD, CLR5, QOutReg5(5 downto 1),
QOutReg6(5 downto 1), QNotOutReg6(5 downto 1));

    Reg7: Register8 port map(CLK, LD, CLR8, LUTSIN(8 downto 1),
QOutReg7(8 downto 1), QNotOutReg7(8 downto 1));

    Reg8: Register8 port map(CLK, LD, CLR8, LUTCOS(8 downto 1),
QOutReg8(8 downto 1), QNotOutReg8(8 downto 1));

```

```

    Shift1: GainShifter port map (QOutReg6(4 downto 1),QOutReg7(8
downto 1),OutShiftSIN(13 downto 1));

    Shift2: GainShifter port map (QOutReg6(4 downto 1),QOutReg8(8
downto 1),OutShiftCOS(13 downto 1));

    Reg9: Register13 port map(CLK, LD, CLR13, OutShiftSIN(13 downto
1), QOutReg9(13 downto 1), QNotOutReg9(13 downto 1));

    Reg10: Register13 port map(CLK, LD, CLR13, OutShiftCOS(13 downto
1), QOutReg10(13 downto 1), QNotOutReg10(13 downto 1));

    Reg11: Register17 port map(CLK, LD, Reset_Inact, OtherBinData-
SIN(17 downto 1), QOutReg11(17 downto 1), QNotOutReg11(17 downto 1));

    Reg12: Register17 port map(CLK, LD, Reset_Inact, OtherBinData-
COS(17 downto 1), QOutReg12(17 downto 1), QNotOutReg12(17 downto 1));

    Add2: Adder16 port map (InputAdder2, QOutReg11(16 downto 1), Ci,
QOutReg11(17), OutAdd2(17 downto 1),Co2);

    Add3: Adder16 port map (InputAdder3, QOutReg12(16 downto 1), Ci,
QOutReg12(17), OutAdd3(17 downto 1),Co3);

    Reg13: Register17 port map(CLK, LD, CLR17, OutAdd2(17 downto 1),
QOutReg13(17 downto 1), QNotOutReg13(17 downto 1));

    Reg14: Register17 port map(CLK, LD, CLR17, OutAdd3(17 downto 1),
QOutReg14(17 downto 1), QNotOutReg14(17 downto 1));

    Control: ControlLogic port map (ODVin, URB, PSVin, CLK, OPER,
CLR13, CLR17, ODVout, PSVout);

```

```

    InputAdder2(13 downto 1) <= QOutReg9(13 downto 1);
    InputAdder2(14) <= QOutReg9(13);
    InputAdder2(15) <= QOutReg9(13);
    InputAdder2(16) <= QOutReg9(13);
    InputAdder3(13 downto 1) <= QOutReg10(13 downto 1);
    InputAdder3(14) <= QOutReg10(13);
    InputAdder3(15) <= QOutReg10(13);
    InputAdder3(16) <= QOutReg10(13);
    DRFM(5 downto 1) <= QOutReg3(5 downto 1);
    Q <= QOutReg13;
    I <= QOutReg14;

```

```

end OneBin;

```

```

entity FourBin is

```

```

    port (Data, Signals: in bit_vector (63 downto 0);
    Output:out bit_vector (63 downto 0); CLK: in bit);
end FourBin;

```

architecture FourBin of FourBin is

component OneBin is

```
    port (PhaseSamp, PhaseInc: in bit_vector (5 downto 1);
          Gain: in bit_vector (4 downto 1);
          ODVin, PSVin: in bit;
          OtherBinDataSIN, OtherBinDataCOS: in bit_vector (17 downto
1);
          Q, I: out bit_vector (17 downto 1);
          ODVout, PSVout: out bit;
          DRFM: out bit_vector (5 downto 1);
          CLK: in bit);
```

end component;

```
signal Q, I, Q1,I1,Q2,I2,Q3,I3, OtherBinDataSIN, OtherBinDataCOS:
bit_vector (17 downto 1);
```

```
signal DRFM0, DRFM1,DRFM2,DRFM3: bit_vector (5 downto 1);
```

```
signal PSVin, ODVin, ODVout0, ODVout1, ODVout2, ODVout3, PSVout0,
PSVout1, PSVout2, PSVout3: bit;
```

```
signal Gain0, Gain1, Gain2, Gain3: bit_vector (4 downto 1);
```

```
signal PhaseInc0, PhaseInc1, PhaseInc2, PhaseInc3, PhaseSamp:
bit_vector (5 downto 1);
```

```
signal URB: bit_vector (2 downto 1);
```

begin -- BIN0 is the primary output

--Data: Bit 63-41 not used,

--40-37 ampscal[0], 36-33 ampscal[1], 32-29 ampscal[2], 28-25 amp-
scal[3],

--24-20 phzinmdat[0], 19-15 phzinmdat[1],14-10 phzinmdat[2], 9-5
phzinmdat[3], 4-0 phasesample

--Signals:Bits 63-38 not used,

--37 PSVin, 36 ODVin, 35-34 URB,

--33-17 OtherBinDataSIN,16-0 OtherBinDataCOS

```
Gain0 <= Data(40 downto 37);
```

```
Gain1 <= Data(36 downto 33);
```

```

Gain2 <= Data(32 downto 29);
Gain3 <= Data(28 downto 25);
PhaseInc0 <= Data(24 downto 20);
PhaseInc1 <= Data(19 downto 15);
PhaseInc2 <= Data(14 downto 10);
PhaseInc3 <= Data(9 downto 5);
PhaseSamp <= Data(4 downto 0);
PSVin <= Signals(37);
ODVin <= Signals(36);
URB <= Signals(35 downto 34);
OtherBinDataSIN <= Signals(33 downto 17);
OtherBinDataCOS <= Signals(16 downto 0);

BIN0: OneBin port map (DRFM1, PhaseInc0, Gain0, ODVout1, PSVout1,
Q1, I1, Q, I, ODVout0, PSVout0, DRFM0, CLK);
BIN1: OneBin port map (DRFM2, PhaseInc1, Gain1, ODVout2, PSVout2, Q2,
I2, Q1, I1, ODVout1, PSVout1, DRFM1, CLK);
BIN2: OneBin port map (DRFM3, PhaseInc2, Gain2, ODVout3, PSVout3, Q3,
I3, Q2, I2, ODVout2, PSVout2, DRFM2, CLK);
BIN3: OneBin port map (PhaseSamp, PhaseInc3, Gain3, ODVin, PSVin,
OtherBinDataSIN, OtherBinDataCOS, Q3, I3, ODVout3, PSVout3, DRFM3,
CLK);

Output(40)<=PSVout0;
Output(39)<=ODVout0;
Output(38 downto 22)<=Q;
Output(21 downto 5)<=I;
Output(4 downto 0)<=DRFM0;
Output(63 downto 41)<="000000000000000000000000";

end FourBin;

```

B. MAKEFILE

```

# -----
# User defines FILES, MAPFILES, and BIN here
# -----
FILES          = main.c
MAPFILES       = FourBinS.mc
BIN            = FourBinTest
# -----

```

```

# User defined macros info supplied here
#
# (Leave commented out if not used)
# -----
MACROS      = my_macro/fourbin.vhd
MY_BLKBOX   = my_macro/fourbin.box
MY_NGO_DIR  = my_macro
MY_INFO     = my_macro/fourbin.info
# -----
# User supplied MCC and MFTN flags
# -----
MY_MCCFLAGS      =
MY_MFTNFLAGS     =
# -----
# User supplied flags for C & Fortran compilers
# -----
CC      = icc # icc for Intel cc for Gnu
FC      = ifc # ifc for Intel f77 for Gnu
LD      = ifc # ifc for Intel cc  for Gnu
MY_CFLAGS =
MY_FFLAGS =
# -----
# No modifications are required below
# -----
MAKIN    ?= $(MC_ROOT)/opt/src/ci/comp/lib/AppRules.make
include $(MAKIN)

```

C. MACRO INFO FILE

```

BEGIN_DEF "Four_Bin"
    MACRO = "FourBin";
    STATEFUL = NO;
    EXTERNAL = NO;
    PIPELINED = YES;
    LATENCY = 9;

    INPUTS = 2:
    I0 = INT 64 BITS (Data[63:0])

```

```

        I1 = INT 64 BITS (Signals[63:0])
        ;

        OUTPUTS = 1:
        O0 = INT 64 BITS (Output[63:0])
        ;

        IN_SIGNAL : 1 BITS "CLK"="CLOCK";

END_DEF

```

D. MACRO BLACKBOX FILE

```

module FourBin (Data, Signals, Output, CLK) /* synthesis syn_black_box
*/ ;

    input [63:0] Data;
    input [63:0] Signals;
    output [63:0] Output;
    input CLK;

endmodule

```

E. C DRIVER PROGRAM

```

/* main.c */

#include <stdio.h>
#include <sys/types.h>
#include <libmap.h>

#define SAMPLE_MAX 500000 /* Maximum number of phase samples. */
#define PADDING 5 /* number of padding sets before and after the sam-
ples */

void FourBinS();
void *Cache_Aligned_Allocate();
void Cache_Aligned_Free();

int main () {

    int i, nmap, mapnum, numofsamps, nbytes;

```

```

short phzsampdat[SAMPLE_MAX];
FILE *fileptr;
long I0, Q0, OtherBinDataSIN, OtherBinDataCOS;
char phzincdat[4], ampscaldat[4];
char PSVin, ODVin, ODVout0, PSVout0, DRFM0, binnumber, URB;
long long temp, binprogram;
long long* dataa;
long long* datab;
long long* datac;

/* Timing variables. */
double tstart, tend, tcume, tttotal;
extern double second();

/* initialization */
tstart = second();
mapnum = 0;
nmap = 1;
OtherBinDataSIN=0;
OtherBinDataCOS=0;
ODVin=0;
numofsamps=0;

/* Read in phase increment values. */
if ((fileptr = fopen("datafiles/phzinc.txt", "r")) == NULL)
    fprintf(stderr, "\n\nTERMINAL FAULT: File phzinc.txt not
found.\n\n");
binnumber = 0;
while (fscanf(fileptr, "%x", &phzincdat[binnumber]) != EOF)
{
    binnumber++;
}
fclose(fileptr);

/* Read in amplitude scaling values */
if ((fileptr = fopen("datafiles/ampscal.txt", "r")) == NULL)
    fprintf(stderr, "\n\nTERMINAL FAULT: File ampscal.txt not
found.\n\n");

```

```

    binnumber = 0;
    while (fscanf(fileptr, "%x", &amp;scaldat[binnumber]) != EOF)
    {
        binnumber++;
    }
    fclose(fileptr);

/* Read in pulse phase samples */
    if ((fileptr = fopen("datafiles/phzsamp.txt", "r")) == NULL)
        fprintf(stderr, "\n\nTERMINAL FAULT: File phzsamp.txt not
found.\n\n");
    numofsamps = 0;
    while (fscanf(fileptr, "%x", &phzsampdat[numofsamps]) != EOF)
    {
        numofsamps++;
    }
    fclose(fileptr);

    tend = second();
    tcume = tend - tstart;
    tttotal = tcume;
    printf ("\n Number of input samples: %d", numofsamps);
    printf ("\n Time for disk access of input data:  %19.10f", tcume);

    tstart = second();
    nbytes = (((numofsamps+PADDING*2+3)/4)*4)*8;
    dataa=Cache_Aligned_Allocate(nbytes);
    datab=Cache_Aligned_Allocate(nbytes);
    datac=Cache_Aligned_Allocate(nbytes);

    tend = second();
    tcume = tend - tstart;
    tttotal = tttotal + tcume;
    printf ("\n Time to allocate the data caches for the MAP:
%19.10f", tcume);

    tstart = second();
/* pack the data as follows:

```

```

Data: Bit 63-41 not used,
      40-37 ampscal[0], 36-33 ampscal[1], 32-29 ampscal[2], 28-25 ampscal[3],
      24-20 phzinmdat[0], 19-15 phzinmdat[1], 14-10 phzinmdat[2], 9-5
      phzinmdat[3], 4-0 phzsampdat

Signals: Bits 63-38 not used,
      37 PSVin, 36 ODVin, 35-34 URB,
      33-17 OtherBinDataSIN, 16-0 OtherBinDataCOS
      pad the data with sets of zero inputs before and after the real
data */

      temp=0;
      temp=((long long) ampscaldat[0] & 0xFLL);
      temp=temp<<4 | ((long long) ampscaldat[1] & 0xFLL);
      temp=temp<<4 | ((long long) ampscaldat[2] & 0xFLL);
      temp=temp<<4 | ((long long) ampscaldat[3] & 0xFLL);
      temp=temp<<5 | ((long long) phzinmdat[0] & 0x1FLL);
      temp=temp<<5 | ((long long) phzinmdat[1] & 0x1FLL);
      temp=temp<<5 | ((long long) phzinmdat[2] & 0x1FLL);
      binprogram=temp<<5 | ((long long) phzinmdat[3] & 0x1FLL);

for (i=0; i<PADDING; i++){
    dataa[i]=binprogram <<5;
    dataa[i+numofsamps+PADDING]=binprogram <<5;
    datab[i]=0;
    datab[i+numofsamps+PADDING]=0;
}
PSVin=1;
URB=3;/*use all 4 rangebins (macro currently ignores this)*/
for (i = 0; i < numofsamps; i++) {
    dataa[i+PADDING]=binprogram<<5 | ((long long) phzsampdat[i]
& 0x1FLL);
    temp=0;
    temp=((long long) PSVin & 0x1LL);
    temp=temp<<1 | ((long long) ODVin & 0x1LL);
    temp=temp<<2 | ((long long) URB & 0x3LL);
    temp=temp<<17 | ((long long) OtherBinDataSIN & 0x1FFFFLL);
    datab[i+PADDING]=temp<<17 | ((long long) OtherBinDataCOS &
0x1FFFFLL);

```

```

    }
    tend = second();
    tcume = tend - tstart;
    tttotal = tttotal + tcume;
    printf ("\n Time to pack the data for transfer to MAP:  %19.10f",
tcume);

    tstart = second();
/* allocate map to this problem */
    if (map_allocate (nmap)) {
        fprintf (stdout, "Map allocation failed.\n");
        exit (1);
    }
    tend = second();
    tcume = tend - tstart;
    tttotal = tttotal + tcume;
    printf ("\n Time for MAP allocation:  %19.10f", tcume);

    tstart = second();
/* call compute */
    FourBinsS (numofsamps+PADDING*2, dataa, datab, datac, mapnum);
    tend = second();
    tcume = tend - tstart;
    tttotal = tttotal + tcume;
    printf ("\n Time for MAP call:  %19.10f", tcume);

    tstart = second();
/* Open output file for writing. */
    if ((fileptr = fopen("datafiles/IandQout.txt", "w")) == NULL)
        fprintf(stderr, "\n\nTERMINAL FAULT:  File
IandQout.txt cannot be written.\n\n");

/* put headers in output file */
    fprintf(fileptr, "Iout  Qout  ODVout  PSVout  DRFM\n");
    fprintf(fileptr, "-----  -----  -----  -----  ----- \n");

/* unpack the results and send to output*/
    for (i = 0; i < numofsamps+PADDING*2; i++) {

```

```

        DRFM0=dataac[i] & 0x1FLL;
        I0=dataac[i]>>5 & 0x1FFFFLL;
        Q0=dataac[i]>>22 & 0x1FFFFLL;
        ODVout0=dataac[i]>>39 & 0x1LL;
        PSVout0=dataac[i]>>40 & 0x1LL;
        fprintf(fileptr, "%05X %05X %01X %01X %02X\n",
I0, Q0, ODVout0, PSVout0, DRFM0);
    }
    fclose (fileptr);

    tend = second();
    tcume = tend - tstart;
    tttotal = tttotal + tcume;
    printf ("\n Time to unpack results and send to output file:
%19.10f", tcume);
    tstart = second();
/* free the map */
    if (map_free (nmap)) {
        printf ("Map deallocation failed. \n");
        exit (1);
    }
    tend = second();
    tcume = tend - tstart;
    tttotal = tttotal + tcume;
    printf ("\n Time to free the MAP: %19.10f", tcume);

    tstart = second();

    Cache_Aligned_Free((char *)dataa);
    Cache_Aligned_Free((char *)datab);
    Cache_Aligned_Free((char *)datac);

    tend = second();
    tcume = tend - tstart;
    tttotal = tttotal + tcume;
    printf ("\n Time to free the data arrays: %19.10f", tcume);
    printf ("\n Total Time: %19.10f\n\n", tttotal);
}

```

F. MAP CODE FILE

```
/* FourBinS.mc */
#include <libmap.h>
#define IBANK MAX_OBM_SIZE
void FourBinS ( int n, long long a[], long long b[], long long c[], int
mapno)
{
    struct {
        long long al[IBANK];
    } banka;
    struct {
        long long bl[IBANK];
    } bankb;
    struct {
        long long cl[IBANK];
    } bankc;

    long long *al      = banka.al;
    long long *bl      = bankb.bl;
    long long *cl      = bankc.cl;

    int i, nbytes;
    /* nbytes = n*8; */
    nbytes = (((n+3)/4)*4)*8;

    cm2obm_a(al, a, nbytes);
    wait_server_a();
    cm2obm_b(bl, b, nbytes);
    wait_server_b();

    for (i = 0; i < n; i++) {
        Four_Bin(al[i], bl[i], &cl[i]);
    }

    obm2cm_c(c, cl, nbytes);
    wait_server_c();
}
```

G. SAMPLE PHASE SAMPLE INPUT FILE

00
01
02
03
04
05
06
07
08
09
0A
0B
0C
0D
0E
0F
10
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F

H. SAMPLE RANGE BIN GAIN INPUT FILE

2
1

2
1

I. SAMPLE SCREEN OUTPUT

Number of input samples: 32

Time for disk access of input data:	0.0002677690
Time to allocate the data caches for the MAP:	0.0000318932
Time to pack the data for transfer to MAP:	0.0000015922
Time for MAP allocation:	0.5351942658
Time for MAP call:	0.0960569127
Time to unpack results and send to output file:	0.0005680144
Time to free the MAP:	1.0062198973
Time to free the data arrays:	0.0000037104
Total Time:	1.6383440550

J. SAMPLE OUTPUT DATA FILE

Iout	Qout	ODVout	PSVout	DRFM
-----	-----	-----	-----	-----
00000	00000	0	1	00
00000	00000	0	1	01
00000	00000	0	1	02
00000	00000	0	1	03
0000F	0FFFC	1	1	04
00007	0FFFE	1	1	05
00016	0FFFB	1	1	06
0000E	0FFFF	1	1	07
0000E	00001	1	1	08
0000D	00005	1	1	09
0000B	00007	1	1	0A
0000A	0000A	1	1	0B
00007	0000C	1	1	0C
00005	0000D	1	1	0D
00002	0000E	1	1	0E
0FFFE	0000E	1	1	0F
0FFFB	0000E	1	1	10
0FFF7	0000D	1	1	11
0FFF5	0000B	1	1	12
0FFF2	0000A	1	1	13

0FFF0	00007	1	1	14
0FFEF	00005	1	1	15
0FFEE	00002	1	1	16
0FFEE	0FFFE	1	1	17
0FFEE	0FFFB	1	1	18
0FFEF	0FFF7	1	1	19
0FFF1	0FFF5	1	1	1A
0FFF2	0FFF2	1	1	1B
0FFF6	0FFF0	1	1	1C
0FFF8	0FFEF	1	1	1D
0FFFB	0FFEE	1	1	1E
0FFFF	0FFEE	1	1	1F
00001	0FFEE	1	0	00
00005	0FFEF	1	0	00
00007	0FFF1	1	0	00
0000A	0FFF2	1	0	00
0FFFD	0FFFA	1	0	00
00006	0FFFA	1	0	00
0FFF8	00000	1	0	00
00000	00000	0	0	00
00000	00000	0	0	00
00000	00000	0	0	00

K. SAMPLE RANGE BIN PHASE ROTATION INPUT FILE

1F

11

1F

11

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D

This appendix contains the raw data collected from the three data sources. The data was edited for format and presentation only by merging multiple data files and screen outputs into single text files.

A. SRC-6E MACRO DATA

Number of input samples: 32

Time for disk access of input data:	0.0002677690
Time to allocate the data caches for the MAP:	0.0000318932
Time to pack the data for transfer to MAP:	0.0000015922
Time for MAP allocation:	0.5351942658
Time for MAP call:	0.0960569127
Time to unpack results and send to output file:	0.0005680144
Time to free the MAP:	1.0062198973
Time to free the data arrays:	0.0000037104
Total Time:	1.6383440550

Number of input samples: 32

Time for disk access of input data:	0.0002689050
Time to allocate the data caches for the MAP:	0.0000345652
Time to pack the data for transfer to MAP:	0.0000015071
Time for MAP allocation:	0.5559181052
Time for MAP call:	0.0958363668
Time to unpack results and send to output file:	0.0003943340
Time to free the MAP:	1.0063609813
Time to free the data arrays:	0.0000037825
Total Time:	1.6588185471

Number of input samples: 32

Time for disk access of input data:	0.0002782845
Time to allocate the data caches for the MAP:	0.0000353730
Time to pack the data for transfer to MAP:	0.0000015228
Time for MAP allocation:	0.5466191977
Time for MAP call:	0.0957191453
Time to unpack results and send to output file:	0.0003934382

Time to free the MAP: 1.0064690704
Time to free the data arrays: 0.0000035836
Total Time: 1.6495196156

Number of input samples: 32

Time for disk access of input data: 0.0002708402
Time to allocate the data caches for the MAP: 0.0000354869
Time to pack the data for transfer to MAP: 0.0000013993
Time for MAP allocation: 0.5923142628
Time for MAP call: 0.0958587045
Time to unpack results and send to output file: 0.0003693907
Time to free the MAP: 1.0066480303
Time to free the data arrays: 0.0000032406
Total Time: 1.6955013552

Number of input samples: 32

Time for disk access of input data: 0.0002633170
Time to allocate the data caches for the MAP: 0.0000347017
Time to pack the data for transfer to MAP: 0.0000012340
Time for MAP allocation: 0.5432007360
Time for MAP call: 0.0957816167
Time to unpack results and send to output file: 0.0003904607
Time to free the MAP: 1.0066382354
Time to free the data arrays: 0.0000032465
Total Time: 1.6463135479

Number of input samples: 64

Time for disk access of input data: 0.0003175845
Time to allocate the data caches for the MAP: 0.0000341569
Time to pack the data for transfer to MAP: 0.0000017385
Time for MAP allocation: 0.5683862600
Time for MAP call: 0.0956988467
Time to unpack results and send to output file: 0.0004762942
Time to free the MAP: 1.0063969221
Time to free the data arrays: 0.0000034788
Total Time: 1.6713152818

Number of input samples: 64
Time for disk access of input data: 0.0003063767
Time to allocate the data caches for the MAP: 0.0000392713
Time to pack the data for transfer to MAP: 0.0000017731
Time for MAP allocation: 0.5590026660
Time for MAP call: 0.0955597495
Time to unpack results and send to output file: 0.0004758422
Time to free the MAP: 1.0067611845
Time to free the data arrays: 0.0000035916
Total Time: 1.6621504548

Number of input samples: 64
Time for disk access of input data: 0.0002826070
Time to allocate the data caches for the MAP: 0.0000367635
Time to pack the data for transfer to MAP: 0.0000017988
Time for MAP allocation: 0.5498394886
Time for MAP call: 0.0957414989
Time to unpack results and send to output file: 0.0004572148
Time to free the MAP: 1.0064087044
Time to free the data arrays: 0.0000038783
Total Time: 1.6527719543

Number of input samples: 64
Time for disk access of input data: 0.0002825416
Time to allocate the data caches for the MAP: 0.0000316966
Time to pack the data for transfer to MAP: 0.0000017237
Time for MAP allocation: 0.5604036079
Time for MAP call: 0.0956780317
Time to unpack results and send to output file: 0.0004765750
Time to free the MAP: 1.0067241372
Time to free the data arrays: 0.0000034452
Total Time: 1.6636017589

Number of input samples: 64
Time for disk access of input data: 0.0002861433
Time to allocate the data caches for the MAP: 0.0000339728
Time to pack the data for transfer to MAP: 0.0000015644

Time for MAP allocation: 0.5389705671
Time for MAP call: 0.0957993332
Time to unpack results and send to output file: 0.0004734413
Time to free the MAP: 1.0063321409
Time to free the data arrays: 0.0000034946
Total Time: 1.6419006575

Number of input samples: 128

Time for disk access of input data: 0.0003499791
Time to allocate the data caches for the MAP: 0.0000348569
Time to pack the data for transfer to MAP: 0.0000021587
Time for MAP allocation: 0.5492326826
Time for MAP call: 0.0956268387
Time to unpack results and send to output file: 0.0006270098
Time to free the MAP: 1.0063932453
Time to free the data arrays: 0.0000038566
Total Time: 1.6522706278

Number of input samples: 128

Time for disk access of input data: 0.0003540977
Time to allocate the data caches for the MAP: 0.0000365776
Time to pack the data for transfer to MAP: 0.0000021408
Time for MAP allocation: 0.5569221765
Time for MAP call: 0.0958692052
Time to unpack results and send to output file: 0.0006243340
Time to free the MAP: 1.0060996530
Time to free the data arrays: 0.0000032404
Total Time: 1.6599114252

Number of input samples: 128

Time for disk access of input data: 0.0003482387
Time to allocate the data caches for the MAP: 0.0000330829
Time to pack the data for transfer to MAP: 0.0000027293
Time for MAP allocation: 0.5580912070
Time for MAP call: 0.0958268696
Time to unpack results and send to output file: 0.0006249647
Time to free the MAP: 1.0061637045

Time to free the data arrays: 0.0000035729
Total Time: 1.6610943696

Number of input samples: 128

Time for disk access of input data: 0.0003452869
Time to allocate the data caches for the MAP: 0.0000329286
Time to pack the data for transfer to MAP: 0.0000021903
Time for MAP allocation: 0.5476320571
Time for MAP call: 0.0957660111
Time to unpack results and send to output file: 0.0006117751
Time to free the MAP: 1.0065463808
Time to free the data arrays: 0.0000031959
Total Time: 1.6509398259

Number of input samples: 128

Time for disk access of input data: 0.0003497200
Time to allocate the data caches for the MAP: 0.0000313107
Time to pack the data for transfer to MAP: 0.0000022150
Time for MAP allocation: 0.6261321505
Time for MAP call: 0.0957161797
Time to unpack results and send to output file: 0.0006258458
Time to free the MAP: 1.0062361960
Time to free the data arrays: 0.0000034363
Total Time: 1.7290970540

Number of input samples: 256

Time for disk access of input data: 0.0004625537
Time to allocate the data caches for the MAP: 0.0000414161
Time to pack the data for transfer to MAP: 0.0000030873
Time for MAP allocation: 0.5701080822
Time for MAP call: 0.0957897472
Time to unpack results and send to output file: 0.0009789201
Time to free the MAP: 1.0057789229
Time to free the data arrays: 0.0000029706
Total Time: 1.6731657000

Number of input samples: 256

Time for disk access of input data: 0.0004759619
 Time to allocate the data caches for the MAP: 0.0000424247
 Time to pack the data for transfer to MAP: 0.0000031962
 Time for MAP allocation: 0.5605598434
 Time for MAP call: 0.0958398278
 Time to unpack results and send to output file: 0.0010340470
 Time to free the MAP: 1.0060577265
 Time to free the data arrays: 0.0000033858
 Total Time: 1.6640164134

Number of input samples: 256

Time for disk access of input data: 0.0004909334
 Time to allocate the data caches for the MAP: 0.0000418077
 Time to pack the data for transfer to MAP: 0.0000032721
 Time for MAP allocation: 0.5610595126
 Time for MAP call: 0.0957838385
 Time to unpack results and send to output file: 0.0010015783
 Time to free the MAP: 1.0060574814
 Time to free the data arrays: 0.0000032188
 Total Time: 1.6644416428

Number of input samples: 256

Time for disk access of input data: 0.0004558661
 Time to allocate the data caches for the MAP: 0.0000427304
 Time to pack the data for transfer to MAP: 0.0000030734
 Time for MAP allocation: 0.5659454288
 Time for MAP call: 0.0957831108
 Time to unpack results and send to output file: 0.0010010364
 Time to free the MAP: 1.0060677676
 Time to free the data arrays: 0.0000037963
 Total Time: 1.6693028099

Number of input samples: 256

Time for disk access of input data: 0.0004885800
 Time to allocate the data caches for the MAP: 0.0000416851
 Time to pack the data for transfer to MAP: 0.0000031890
 Time for MAP allocation: 0.5511499354

Time for MAP call: 0.0961975577
Time to unpack results and send to output file: 0.0010007792
Time to free the MAP: 1.0055322002
Time to free the data arrays: 0.0000034818
Total Time: 1.6544174084

Number of input samples: 512

Time for disk access of input data: 0.0006995983
Time to allocate the data caches for the MAP: 0.0000495407
Time to pack the data for transfer to MAP: 0.0000059341
Time for MAP allocation: 0.5635743956
Time for MAP call: 0.0958592651
Time to unpack results and send to output file: 0.0017114796
Time to free the MAP: 1.0053390195
Time to free the data arrays: 0.0000028657
Total Time: 1.6672420986

Number of input samples: 512

Time for disk access of input data: 0.0007183850
Time to allocate the data caches for the MAP: 0.0000513633
Time to pack the data for transfer to MAP: 0.0000054903
Time for MAP allocation: 0.5580191022
Time for MAP call: 0.0958093556
Time to unpack results and send to output file: 0.0017153421
Time to free the MAP: 1.0050138898
Time to free the data arrays: 0.0000028241
Total Time: 1.6613357524

Number of input samples: 512

Time for disk access of input data: 0.0007295769
Time to allocate the data caches for the MAP: 0.0000498671
Time to pack the data for transfer to MAP: 0.0000053103
Time for MAP allocation: 0.5441085579
Time for MAP call: 0.0957256651
Time to unpack results and send to output file: 0.0017012210
Time to free the MAP: 1.0051159665
Time to free the data arrays: 0.0000031229

Total Time: 1.6474392877

Number of input samples: 512

Time for disk access of input data: 0.0007316072
Time to allocate the data caches for the MAP: 0.0000485241
Time to pack the data for transfer to MAP: 0.0000066572
Time for MAP allocation: 0.5413774263
Time for MAP call: 0.0957333604
Time to unpack results and send to output file: 0.0017187052
Time to free the MAP: 1.0052396681
Time to free the data arrays: 0.0000030271
Total Time: 1.6448589756

Number of input samples: 512

Time for disk access of input data: 0.0007073195
Time to allocate the data caches for the MAP: 0.0000512257
Time to pack the data for transfer to MAP: 0.0000051294
Time for MAP allocation: 0.5688409651
Time for MAP call: 0.0961722968
Time to unpack results and send to output file: 0.0017124555
Time to free the MAP: 1.0049554719
Time to free the data arrays: 0.0000037834
Total Time: 1.6724486473

Number of input samples: 1024

Time for disk access of input data: 0.0011598321
Time to allocate the data caches for the MAP: 0.0000747768
Time to pack the data for transfer to MAP: 0.0000107293
Time for MAP allocation: 0.5627898064
Time for MAP call: 0.0958103127
Time to unpack results and send to output file: 0.0031125588
Time to free the MAP: 1.0039292726
Time to free the data arrays: 0.0000034244
Total Time: 1.6668907131

Number of input samples: 1024

Time for disk access of input data: 0.0011944289

Time to allocate the data caches for the MAP: 0.0000739026
 Time to pack the data for transfer to MAP: 0.0000107085
 Time for MAP allocation: 0.5614359006
 Time for MAP call: 0.0958402471
 Time to unpack results and send to output file: 0.0031293658
 Time to free the MAP: 1.0033827050
 Time to free the data arrays: 0.0000034058
 Total Time: 1.6650706644

Number of input samples: 1024

Time for disk access of input data: 0.0012244619
 Time to allocate the data caches for the MAP: 0.0000784731
 Time to pack the data for transfer to MAP: 0.0000101171
 Time for MAP allocation: 0.5669450610
 Time for MAP call: 0.0956240124
 Time to unpack results and send to output file: 0.0031523541
 Time to free the MAP: 1.0037887276
 Time to free the data arrays: 0.0000034323
 Total Time: 1.6708266397

Number of input samples: 1024

Time for disk access of input data: 0.0011605509
 Time to allocate the data caches for the MAP: 0.0000752337
 Time to pack the data for transfer to MAP: 0.0000101676
 Time for MAP allocation: 0.6236788962
 Time for MAP call: 0.0958688937
 Time to unpack results and send to output file: 0.0031311201
 Time to free the MAP: 1.0035382028
 Time to free the data arrays: 0.0000033830
 Total Time: 1.7274664481

Number of input samples: 1024

Time for disk access of input data: 0.0011683335
 Time to allocate the data caches for the MAP: 0.0000749340
 Time to pack the data for transfer to MAP: 0.0000102182
 Time for MAP allocation: 0.5578285100
 Time for MAP call: 0.0958067626

Time to unpack results and send to output file: 0.0031323314
Time to free the MAP: 1.0037003675
Time to free the data arrays: 0.0000037855
Total Time: 1.6617252427

Number of input samples: 2048

Time for disk access of input data: 0.0021121742
Time to allocate the data caches for the MAP: 0.0001252035
Time to pack the data for transfer to MAP: 0.0000210125
Time for MAP allocation: 0.5629239089
Time for MAP call: 0.0960486161
Time to unpack results and send to output file: 0.0059072306
Time to free the MAP: 1.0108930274
Time to free the data arrays: 0.0000037240
Total Time: 1.6780348972

Number of input samples: 2048

Time for disk access of input data: 0.0021581480
Time to allocate the data caches for the MAP: 0.0001240052
Time to pack the data for transfer to MAP: 0.0000198348
Time for MAP allocation: 0.5655415718
Time for MAP call: 0.0959882690
Time to unpack results and send to output file: 0.0059822606
Time to free the MAP: 1.0108668191
Time to free the data arrays: 0.0000034374
Total Time: 1.6806843458

Number of input samples: 2048

Time for disk access of input data: 0.0020997350
Time to allocate the data caches for the MAP: 0.0001187363
Time to pack the data for transfer to MAP: 0.0000202602
Time for MAP allocation: 0.5527371586
Time for MAP call: 0.0959806834
Time to unpack results and send to output file: 0.0060287457
Time to free the MAP: 1.0008449403
Time to free the data arrays: 0.0000035115
Total Time: 1.6578337709

Number of input samples: 2048
Time for disk access of input data: 0.0021319420
Time to allocate the data caches for the MAP: 0.0001291689
Time to pack the data for transfer to MAP: 0.0000213944
Time for MAP allocation: 0.5652251829
Time for MAP call: 0.0962193980
Time to unpack results and send to output file: 0.0060068518
Time to free the MAP: 1.0103367549
Time to free the data arrays: 0.0000033099
Total Time: 1.6800740027

Number of input samples: 2048
Time for disk access of input data: 0.0021120040
Time to allocate the data caches for the MAP: 0.0001270320
Time to pack the data for transfer to MAP: 0.0000202671
Time for MAP allocation: 0.6232019155
Time for MAP call: 0.0960557450
Time to unpack results and send to output file: 0.0059984444
Time to free the MAP: 1.0108003173
Time to free the data arrays: 0.0000030409
Total Time: 1.7383187662

Number of input samples: 4096
Time for disk access of input data: 0.0039900679
Time to allocate the data caches for the MAP: 0.0002096794
Time to pack the data for transfer to MAP: 0.0000459382
Time for MAP allocation: 0.5496688402
Time for MAP call: 0.0962074068
Time to unpack results and send to output file: 0.0116166790
Time to free the MAP: 1.0047562273
Time to free the data arrays: 0.0000039219
Total Time: 1.6664987608

Number of input samples: 4096
Time for disk access of input data: 0.0039976444
Time to allocate the data caches for the MAP: 0.0002221994

Time to pack the data for transfer to MAP: 0.0000597746
 Time for MAP allocation: 0.5453757423
 Time for MAP call: 0.0963799618
 Time to unpack results and send to output file: 0.0116735057
 Time to free the MAP: 1.0045205898
 Time to free the data arrays: 0.0000045923
 Total Time: 1.6622340104

Number of input samples: 4096

Time for disk access of input data: 0.0040123798
 Time to allocate the data caches for the MAP: 0.0002263054
 Time to pack the data for transfer to MAP: 0.0000590547
 Time for MAP allocation: 0.5631937416
 Time for MAP call: 0.0962763444
 Time to unpack results and send to output file: 0.0119604370
 Time to free the MAP: 1.0046578696
 Time to free the data arrays: 0.0000045894
 Total Time: 1.6803907219

Number of input samples: 4096

Time for disk access of input data: 0.0039810075
 Time to allocate the data caches for the MAP: 0.0002196887
 Time to pack the data for transfer to MAP: 0.0000460430
 Time for MAP allocation: 0.5485791007
 Time for MAP call: 0.0962252609
 Time to unpack results and send to output file: 0.0116839264
 Time to free the MAP: 1.0046424461
 Time to free the data arrays: 0.0000035757
 Total Time: 1.6653810490

Number of input samples: 4096

Time for disk access of input data: 0.0040224830
 Time to allocate the data caches for the MAP: 0.0002064347
 Time to pack the data for transfer to MAP: 0.0000547056
 Time for MAP allocation: 0.6224038432
 Time for MAP call: 0.0960545790
 Time to unpack results and send to output file: 0.0116945519

Time to free the MAP: 1.0048462921
Time to free the data arrays: 0.0000038892
Total Time: 1.7392867787

Number of input samples: 8192

Time for disk access of input data: 0.0078223245
Time to allocate the data caches for the MAP: 0.0004023044
Time to pack the data for transfer to MAP: 0.0002119705
Time for MAP allocation: 0.5812084946
Time for MAP call: 0.0964297191
Time to unpack results and send to output file: 0.0228781873
Time to free the MAP: 1.0130009019
Time to free the data arrays: 0.0000038685
Total Time: 1.7219577708

Number of input samples: 8192

Time for disk access of input data: 0.0078307251
Time to allocate the data caches for the MAP: 0.0004436297
Time to pack the data for transfer to MAP: 0.0002067542
Time for MAP allocation: 0.5730418858
Time for MAP call: 0.0966391728
Time to unpack results and send to output file: 0.0230338760
Time to free the MAP: 1.0131350597
Time to free the data arrays: 0.0000040277
Total Time: 1.7143351309

Number of input samples: 8192

Time for disk access of input data: 0.0077393767
Time to allocate the data caches for the MAP: 0.0004243039
Time to pack the data for transfer to MAP: 0.0001817426
Time for MAP allocation: 0.6185239698
Time for MAP call: 0.0966174048
Time to unpack results and send to output file: 0.0230480030
Time to free the MAP: 1.0029756321
Time to free the data arrays: 0.0000042630
Total Time: 1.7495146959

Number of input samples: 8192
 Time for disk access of input data: 0.0078371656
 Time to allocate the data caches for the MAP: 0.0004584440
 Time to pack the data for transfer to MAP: 0.0002532533
 Time for MAP allocation: 0.5599991367
 Time for MAP call: 0.0966641570
 Time to unpack results and send to output file: 0.0232095667
 Time to free the MAP: 1.0027152393
 Time to free the data arrays: 0.0000033759
 Total Time: 1.6911403385

Number of input samples: 8192
 Time for disk access of input data: 0.0077986943
 Time to allocate the data caches for the MAP: 0.0004437948
 Time to pack the data for transfer to MAP: 0.0001976624
 Time for MAP allocation: 0.6172022498
 Time for MAP call: 0.0967147538
 Time to unpack results and send to output file: 0.0237511987
 Time to free the MAP: 1.0120747115
 Time to free the data arrays: 0.0000037410
 Total Time: 1.7581868063

Number of input samples: 16384
 Time for disk access of input data: 0.0154911431
 Time to allocate the data caches for the MAP: 0.0011080426
 Time to pack the data for transfer to MAP: 0.0008457769
 Time for MAP allocation: 0.5440686744
 Time for MAP call: 0.0972487617
 Time to unpack results and send to output file: 0.0456674025
 Time to free the MAP: 1.0098512721
 Time to free the data arrays: 0.0001430044
 Total Time: 1.7144240776

Number of input samples: 16384
 Time for disk access of input data: 0.0155376915
 Time to allocate the data caches for the MAP: 0.0010734378
 Time to pack the data for transfer to MAP: 0.0007945728

Time for MAP allocation: 0.5509032889
Time for MAP call: 0.0973605433
Time to unpack results and send to output file: 0.0459777423
Time to free the MAP: 1.0094799668
Time to free the data arrays: 0.0001454507
Total Time: 1.7212726941

Number of input samples: 16384

Time for disk access of input data: 0.0154718570
Time to allocate the data caches for the MAP: 0.0010781133
Time to pack the data for transfer to MAP: 0.0008486386
Time for MAP allocation: 0.6051188154
Time for MAP call: 0.0972983311
Time to unpack results and send to output file: 0.0458111949
Time to free the MAP: 1.0094288469
Time to free the data arrays: 0.0001662607
Total Time: 1.7752220580

Number of input samples: 16384

Time for disk access of input data: 0.0154333424
Time to allocate the data caches for the MAP: 0.0011713841
Time to pack the data for transfer to MAP: 0.0008559671
Time for MAP allocation: 0.6519303209
Time for MAP call: 0.0973521378
Time to unpack results and send to output file: 0.0459583633
Time to free the MAP: 1.0095304322
Time to free the data arrays: 0.0001494944
Total Time: 1.8223814423

Number of input samples: 16384

Time for disk access of input data: 0.0152250228
Time to allocate the data caches for the MAP: 0.0011885736
Time to pack the data for transfer to MAP: 0.0010387779
Time for MAP allocation: 0.6485815154
Time for MAP call: 0.0975391807
Time to unpack results and send to output file: 0.0464225369
Time to free the MAP: 1.0087604537

Time to free the data arrays: 0.0001434464
Total Time: 1.8188995075

Number of input samples: 32768

Time for disk access of input data: 0.0316404635
Time to allocate the data caches for the MAP: 0.0034781572
Time to pack the data for transfer to MAP: 0.0029248423
Time for MAP allocation: 0.6252940828
Time for MAP call: 0.0988847446
Time to unpack results and send to output file: 0.0913046420
Time to free the MAP: 1.0023308353
Time to free the data arrays: 0.0002494756
Total Time: 1.8561072433

Number of input samples: 32768

Time for disk access of input data: 0.0305634427
Time to allocate the data caches for the MAP: 0.0022265891
Time to pack the data for transfer to MAP: 0.0023904122
Time for MAP allocation: 0.5508623337
Time for MAP call: 0.0992578574
Time to unpack results and send to output file: 0.0917728660
Time to free the MAP: 1.0118414713
Time to free the data arrays: 0.0002532977
Total Time: 1.7891682702

Number of input samples: 32768

Time for disk access of input data: 0.0307901613
Time to allocate the data caches for the MAP: 0.0022666832
Time to pack the data for transfer to MAP: 0.0033268828
Time for MAP allocation: 0.5559343584
Time for MAP call: 0.0989380410
Time to unpack results and send to output file: 0.0916615997
Time to free the MAP: 1.0122256754
Time to free the data arrays: 0.0002460956
Total Time: 1.7953894975

Number of input samples: 32768

Time for disk access of input data: 0.0311453124
 Time to allocate the data caches for the MAP: 0.0022655370
 Time to pack the data for transfer to MAP: 0.0023452306
 Time for MAP allocation: 0.5423111092
 Time for MAP call: 0.0989094210
 Time to unpack results and send to output file: 0.0934043771
 Time to free the MAP: 1.0106256046
 Time to free the data arrays: 0.0002398184
 Total Time: 1.7812464102

Number of input samples: 32768

Time for disk access of input data: 0.0303366007
 Time to allocate the data caches for the MAP: 0.0023124818
 Time to pack the data for transfer to MAP: 0.0021618444
 Time for MAP allocation: 0.5487904255
 Time for MAP call: 0.0987779063
 Time to unpack results and send to output file: 0.0917617551
 Time to free the MAP: 1.0123094293
 Time to free the data arrays: 0.0002469738
 Total Time: 1.7866974169

Number of input samples: 65536

Time for disk access of input data: 0.0642365906
 Time to allocate the data caches for the MAP: 0.0049776484
 Time to pack the data for transfer to MAP: 0.0047179230
 Time for MAP allocation: 0.5696820675
 Time for MAP call: 0.1021540966
 Time to unpack results and send to output file: 0.1828584631
 Time to free the MAP: 1.0077824403
 Time to free the data arrays: 0.0004443910
 Total Time: 1.9368536205

Number of input samples: 65536

Time for disk access of input data: 0.0609015811
 Time to allocate the data caches for the MAP: 0.0075692546
 Time to pack the data for transfer to MAP: 0.0054795219
 Time for MAP allocation: 0.5880060163

Time for MAP call: 0.1020899076
Time to unpack results and send to output file: 0.1833274872
Time to free the MAP: 1.0068394203
Time to free the data arrays: 0.0004423104
Total Time: 1.9546554993

Number of input samples: 65536
Time for disk access of input data: 0.0635116564
Time to allocate the data caches for the MAP: 0.0055077909
Time to pack the data for transfer to MAP: 0.0046081302
Time for MAP allocation: 0.5434199974
Time for MAP call: 0.1022932166
Time to unpack results and send to output file: 0.1835829851
Time to free the MAP: 1.0065619061
Time to free the data arrays: 0.0004432341
Total Time: 1.9099289168

Number of input samples: 65536
Time for disk access of input data: 0.0602645045
Time to allocate the data caches for the MAP: 0.0064083445
Time to pack the data for transfer to MAP: 0.0063568686
Time for MAP allocation: 0.5564223806
Time for MAP call: 0.1022868127
Time to unpack results and send to output file: 0.1841379679
Time to free the MAP: 1.0061306859
Time to free the data arrays: 0.0004438511
Total Time: 1.9224514159

Number of input samples: 65536
Time for disk access of input data: 0.0603332552
Time to allocate the data caches for the MAP: 0.0076104885
Time to pack the data for transfer to MAP: 0.0053092038
Time for MAP allocation: 0.5615856629
Time for MAP call: 0.1317872155
Time to unpack results and send to output file: 0.1840570403
Time to free the MAP: 1.0066493416
Time to free the data arrays: 0.0004394368

Total Time: 1.9577716446

Number of input samples: 131072

Time for disk access of input data: 0.1220448297
Time to allocate the data caches for the MAP: 0.0093876077
Time to pack the data for transfer to MAP: 0.0089142388
Time for MAP allocation: 0.7330934388
Time for MAP call: 0.1090398467
Time to unpack results and send to output file: 0.6348346174
Time to free the MAP: 1.0086661741
Time to free the data arrays: 0.0009079930
Total Time: 2.6268887462

Number of input samples: 131072

Time for disk access of input data: 0.1201003324
Time to allocate the data caches for the MAP: 0.0093825150
Time to pack the data for transfer to MAP: 0.0089177907
Time for MAP allocation: 0.5360358666
Time for MAP call: 0.1088606189
Time to unpack results and send to output file: 0.5496880324
Time to free the MAP: 1.0040097385
Time to free the data arrays: 0.0009155073
Total Time: 2.3379104019

Number of input samples: 131072

Time for disk access of input data: 0.1229936327
Time to allocate the data caches for the MAP: 0.0093958263
Time to pack the data for transfer to MAP: 0.0088779075
Time for MAP allocation: 0.5349584831
Time for MAP call: 0.1089405220
Time to unpack results and send to output file: 0.6081954039
Time to free the MAP: 1.0055497212
Time to free the data arrays: 0.0009038526
Total Time: 2.3998153493

Number of input samples: 131072

Time for disk access of input data: 0.1227266786

Time to allocate the data caches for the MAP: 0.0094592113
 Time to pack the data for transfer to MAP: 0.0088770638
 Time for MAP allocation: 0.5336023561
 Time for MAP call: 0.1086331309
 Time to unpack results and send to output file: 0.6763001675
 Time to free the MAP: 1.0175269860
 Time to free the data arrays: 0.0009064127
 Total Time: 2.4780320070

Number of input samples: 131072

Time for disk access of input data: 0.1200560762
 Time to allocate the data caches for the MAP: 0.0093610427
 Time to pack the data for transfer to MAP: 0.0089234996
 Time for MAP allocation: 0.5939230269
 Time for MAP call: 0.1088982179
 Time to unpack results and send to output file: 0.6142954248
 Time to free the MAP: 1.0194287639
 Time to free the data arrays: 0.0009110189
 Total Time: 2.4757970710

Number of input samples: 262144

Time for disk access of input data: 0.2439861309
 Time to allocate the data caches for the MAP: 0.0188529759
 Time to pack the data for transfer to MAP: 0.0178578915
 Time for MAP allocation: 0.6893795544
 Time for MAP call: 0.1218434373
 Time to unpack results and send to output file: 1.1910147560
 Time to free the MAP: 1.0097043914
 Time to free the data arrays: 0.0017815761
 Total Time: 3.2944207134

Number of input samples: 262144

Time for disk access of input data: 0.2439314867
 Time to allocate the data caches for the MAP: 0.0188461833
 Time to pack the data for transfer to MAP: 0.0178297551
 Time for MAP allocation: 0.5352123523
 Time for MAP call: 0.1242827591

Time to unpack results and send to output file: 1.3234282694
Time to free the MAP: 1.0119794117
Time to free the data arrays: 0.0017971567
Total Time: 3.2773073744

Number of input samples: 262144
Time for disk access of input data: 0.2399882292
Time to allocate the data caches for the MAP: 0.0188378134
Time to pack the data for transfer to MAP: 0.0179187362
Time for MAP allocation: 0.5406684882
Time for MAP call: 0.1219761128
Time to unpack results and send to output file: 1.2522615479
Time to free the MAP: 1.0082841585
Time to free the data arrays: 0.0017902603
Total Time: 3.2017253466

Number of input samples: 262144
Time for disk access of input data: 0.2455214649
Time to allocate the data caches for the MAP: 0.0190074293
Time to pack the data for transfer to MAP: 0.0178621577
Time for MAP allocation: 0.5320163383
Time for MAP call: 0.1217802391
Time to unpack results and send to output file: 1.2010226196
Time to free the MAP: 1.0097881788
Time to free the data arrays: 0.0018161935
Total Time: 3.1488146211

Number of input samples: 262144
Time for disk access of input data: 0.2401958352
Time to allocate the data caches for the MAP: 0.0190414892
Time to pack the data for transfer to MAP: 0.0178416384
Time for MAP allocation: 0.5350196342
Time for MAP call: 0.1220957637
Time to unpack results and send to output file: 1.2980060737
Time to free the MAP: 1.0024299308
Time to free the data arrays: 0.0018355162
Total Time: 3.2364658814

Number of input samples: 500000
Time for disk access of input data: 0.4576555924
Time to allocate the data caches for the MAP: 0.0357430988
Time to pack the data for transfer to MAP: 0.0341176347
Time for MAP allocation: 0.5400339917
Time for MAP call: 0.1453687382
Time to unpack results and send to output file: 2.2802877245
Time to free the MAP: 1.0271661955
Time to free the data arrays: 0.0034380277
Total Time: 4.5238110034

Number of input samples: 500000
Time for disk access of input data: 0.4639648099
Time to allocate the data caches for the MAP: 0.0356763230
Time to pack the data for transfer to MAP: 0.0341234227
Time for MAP allocation: 0.5855113962
Time for MAP call: 0.1453061264
Time to unpack results and send to output file: 2.4983290645
Time to free the MAP: 1.0092234109
Time to free the data arrays: 0.0034494809
Total Time: 4.7755840344

Number of input samples: 500000
Time for disk access of input data: 0.4578109535
Time to allocate the data caches for the MAP: 0.0359110724
Time to pack the data for transfer to MAP: 0.0339820569
Time for MAP allocation: 0.5354018536
Time for MAP call: 0.1452250762
Time to unpack results and send to output file: 2.6184729122
Time to free the MAP: 1.0091233435
Time to free the data arrays: 0.0034339475
Total Time: 4.8393612159

Number of input samples: 500000
Time for disk access of input data: 0.4575756039
Time to allocate the data caches for the MAP: 0.0357971735

Time to pack the data for transfer to MAP: 0.0339763491
 Time for MAP allocation: 0.5455715893
 Time for MAP call: 0.1451846370
 Time to unpack results and send to output file: 2.4218357695
 Time to free the MAP: 1.0357822277
 Time to free the data arrays: 0.0033822233
 Total Time: 4.6791055732

Number of input samples: 500000

Time for disk access of input data: 0.4678256030
 Time to allocate the data caches for the MAP: 0.0413532404
 Time to pack the data for transfer to MAP: 0.0363923355
 Time for MAP allocation: 0.5508184513
 Time for MAP call: 0.1455726750
 Time to unpack results and send to output file: 2.2853761229
 Time to free the MAP: 1.0018905374
 Time to free the data arrays: 0.0033834348
 Total Time: 4.5326124002

B. SRC-6E C PROGRAM DATA

Time to complete 32 samples: 0.1400 seconds.
 Time to complete 32 samples: 0.1300 seconds.
 Time to complete 32 samples: 0.1300 seconds.
 Time to complete 32 samples: 0.1300 seconds.
 Time to complete 32 samples: 0.1400 seconds.
 Time to complete 64 samples: 0.1400 seconds.
 Time to complete 64 samples: 0.1200 seconds.
 Time to complete 64 samples: 0.1200 seconds.
 Time to complete 64 samples: 0.1100 seconds.
 Time to complete 64 samples: 0.1200 seconds.
 Time to complete 128 samples: 0.1200 seconds.
 Time to complete 128 samples: 0.1200 seconds.
 Time to complete 128 samples: 0.1200 seconds.
 Time to complete 128 samples: 0.1500 seconds.
 Time to complete 128 samples: 0.1200 seconds.
 Time to complete 256 samples: 0.1200 seconds.
 Time to complete 256 samples: 0.1200 seconds.
 Time to complete 256 samples: 0.1200 seconds.

Time to complete 256 samples: 0.1200 seconds.
Time to complete 256 samples: 0.1200 seconds.
Time to complete 512 samples: 0.1200 seconds.
Time to complete 512 samples: 0.1200 seconds.
Time to complete 512 samples: 0.1200 seconds.
Time to complete 512 samples: 0.1200 seconds.
Time to complete 512 samples: 0.1200 seconds.
Time to complete 1024 samples: 0.1200 seconds.
Time to complete 1024 samples: 0.1200 seconds.
Time to complete 1024 samples: 0.1200 seconds.
Time to complete 1024 samples: 0.1300 seconds.
Time to complete 1024 samples: 0.1200 seconds.
Time to complete 2048 samples: 0.1200 seconds.
Time to complete 2048 samples: 0.1300 seconds.
Time to complete 2048 samples: 0.1300 seconds.
Time to complete 2048 samples: 0.1200 seconds.
Time to complete 2048 samples: 0.1300 seconds.
Time to complete 4096 samples: 0.1300 seconds.
Time to complete 4096 samples: 0.1300 seconds.
Time to complete 4096 samples: 0.1300 seconds.
Time to complete 4096 samples: 0.1400 seconds.
Time to complete 4096 samples: 0.1400 seconds.
Time to complete 8192 samples: 0.1500 seconds.
Time to complete 8192 samples: 0.1500 seconds.
Time to complete 8192 samples: 0.1500 seconds.
Time to complete 8192 samples: 0.1500 seconds.
Time to complete 8192 samples: 0.1500 seconds.
Time to complete 16384 samples: 0.2100 seconds.
Time to complete 16384 samples: 0.1800 seconds.
Time to complete 16384 samples: 0.1600 seconds.
Time to complete 16384 samples: 0.1800 seconds.
Time to complete 16384 samples: 0.1800 seconds.
Time to complete 32768 samples: 0.2600 seconds.
Time to complete 32768 samples: 0.2500 seconds.
Time to complete 32768 samples: 0.2600 seconds.
Time to complete 32768 samples: 0.2600 seconds.
Time to complete 32768 samples: 0.2600 seconds.

Time to complete 65536 samples: 0.3800 seconds.
Time to complete 65536 samples: 0.3500 seconds.
Time to complete 65536 samples: 0.3500 seconds.
Time to complete 65536 samples: 0.3500 seconds.
Time to complete 65536 samples: 0.3700 seconds.
Time to complete 131072 samples: 0.6000 seconds.
Time to complete 131072 samples: 0.6700 seconds.
Time to complete 131072 samples: 0.6600 seconds.
Time to complete 131072 samples: 0.6300 seconds.
Time to complete 131072 samples: 0.6100 seconds.
Time to complete 262144 samples: 1.1200 seconds.
Time to complete 262144 samples: 1.0700 seconds.
Time to complete 262144 samples: 1.2800 seconds.
Time to complete 262144 samples: 1.1300 seconds.
Time to complete 262144 samples: 1.1800 seconds.
Time to complete 500000 samples: 2.0300 seconds.
Time to complete 500000 samples: 2.0400 seconds.
Time to complete 500000 samples: 2.0400 seconds.
Time to complete 500000 samples: 2.0300 seconds.
Time to complete 500000 samples: 2.0700 seconds.

C. WINDOWS C PROGRAM DATA

Time to complete 32 samples: 0.0310 seconds.
Time to complete 32 samples: 0.0460 seconds.
Time to complete 32 samples: 0.0460 seconds.
Time to complete 32 samples: 0.0460 seconds.
Time to complete 32 samples: 0.0460 seconds.
Time to complete 64 samples: 0.0460 seconds.
Time to complete 64 samples: 0.0460 seconds.
Time to complete 64 samples: 0.0460 seconds.
Time to complete 64 samples: 0.0460 seconds.
Time to complete 128 samples: 0.0310 seconds.
Time to complete 128 samples: 0.0460 seconds.
Time to complete 128 samples: 0.0460 seconds.
Time to complete 128 samples: 0.0460 seconds.
Time to complete 128 samples: 0.0460 seconds.
Time to complete 256 samples: 0.0460 seconds.

Time to complete 256 samples: 0.0460 seconds.
Time to complete 256 samples: 0.0460 seconds.
Time to complete 256 samples: 0.0620 seconds.
Time to complete 256 samples: 0.0620 seconds.
Time to complete 512 samples: 0.0460 seconds.
Time to complete 512 samples: 0.0460 seconds.
Time to complete 512 samples: 0.0460 seconds.
Time to complete 512 samples: 0.0460 seconds.
Time to complete 512 samples: 0.0620 seconds.
Time to complete 1024 samples: 0.0620 seconds.
Time to complete 1024 samples: 0.0620 seconds.
Time to complete 1024 samples: 0.0620 seconds.
Time to complete 1024 samples: 0.0620 seconds.
Time to complete 1024 samples: 0.0620 seconds.
Time to complete 2048 samples: 0.0620 seconds.
Time to complete 2048 samples: 0.0620 seconds.
Time to complete 2048 samples: 0.0620 seconds.
Time to complete 2048 samples: 0.0620 seconds.
Time to complete 2048 samples: 0.0620 seconds.
Time to complete 4096 samples: 0.0620 seconds.
Time to complete 4096 samples: 0.0620 seconds.
Time to complete 4096 samples: 0.0620 seconds.
Time to complete 4096 samples: 0.0780 seconds.
Time to complete 4096 samples: 0.0780 seconds.
Time to complete 8192 samples: 0.0930 seconds.
Time to complete 8192 samples: 0.0930 seconds.
Time to complete 8192 samples: 0.0930 seconds.
Time to complete 8192 samples: 0.0930 seconds.
Time to complete 8192 samples: 0.0930 seconds.
Time to complete 16384 samples: 0.1240 seconds.
Time to complete 16384 samples: 0.1240 seconds.
Time to complete 16384 samples: 0.1400 seconds.
Time to complete 16384 samples: 0.1400 seconds.
Time to complete 16384 samples: 0.1400 seconds.
Time to complete 32768 samples: 0.2030 seconds.
Time to complete 32768 samples: 0.2180 seconds.
Time to complete 32768 samples: 0.2180 seconds.

Time to complete 32768 samples: 0.2180 seconds.
Time to complete 32768 samples: 0.2180 seconds.
Time to complete 65536 samples: 0.3280 seconds.
Time to complete 65536 samples: 0.3430 seconds.
Time to complete 65536 samples: 0.3430 seconds.
Time to complete 65536 samples: 0.3430 seconds.
Time to complete 65536 samples: 0.4370 seconds.
Time to complete 131072 samples: 0.5780 seconds.
Time to complete 131072 samples: 0.5780 seconds.
Time to complete 131072 samples: 0.5930 seconds.
Time to complete 131072 samples: 0.5930 seconds.
Time to complete 131072 samples: 0.5930 seconds.
Time to complete 262144 samples: 1.0620 seconds.
Time to complete 262144 samples: 1.0780 seconds.
Time to complete 262144 samples: 1.0780 seconds.
Time to complete 262144 samples: 1.0780 seconds.
Time to complete 262144 samples: 1.1240 seconds.
Time to complete 500000 samples: 2.0310 seconds.
Time to complete 500000 samples: 2.0310 seconds.
Time to complete 500000 samples: 2.8900 seconds.
Time to complete 500000 samples: 3.1710 seconds.
Time to complete 500000 samples: 3.1710 seconds.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] David Caliga and David Peter Barker, "Delivering Acceleration: The Potential for Increased HPC Application Performance Using Reconfigurable Logic," ACM 1-58113-293-X/01/0011, November 2001.
- [2] "SRC-6E MAP® Hardware Guide," SRC-005-03, SRC Computers, Inc., Colorado Springs, January 6, 2003.
- [3] "Virtex-II Platform FPGAs: Complete Data Sheet, DC and Switching Characteristics," DS031-3 (v3.1), Xilinx, Inc., San Jose, CA, October 14, 2003. From website: <http://direct.xilinx.com/bvdocs/publications/ds031.pdf>, accessed December 2003.
- [4] "SRC-6E C Programming Environment V1.5 Guide," SRC-007-08, SRC Computers Inc., Colorado Springs, September 5, 2003.
- [5] "SRC-6E Fortran Programming Environment V1.5 Guide," SRC-006-08, SRC Computers Inc., Colorado Springs, September 5, 2003.
- [6] "SRC-6E MAP® Macro Developers Guide," SRC-008-01, SRC Computers Inc., Colorado Springs, September 23, 2002.
- [7] "SRC-6E Programming Environment V1.5 Technical Note: Supported Macros," SRC Computers Inc., Colorado Springs, September 5, 2003.
- [8] Charles H. Roth, Jr., *Digital Systems Design Using VHDL*, PWS Publishing Company, Boston, 1998.
- [9] Author Unknown. Unpublished project notes from previous work. Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Chairman, Code EC
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California
4. Alan Hunsberger
National Security Agency
Ft. Meade, MD
5. Dr. Russell Duren
Baylor University
Engineering Department
Rogers, TX